NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2017; **00**:1–22
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nla

Improving the stability and robustness of incomplete symmetric indefinite factorization preconditioners

Jennifer Scott[1]*, Miroslav Tůma[2]

[1]*Scientific Computing Department, Rutherford Appleton Laboratory, Didcot, Oxfordshire, OX11 0QX, UK. E-mail: jennifer.scott@stfc.ac.uk*
[2]*Department of Numerical Mathematics, Faculty of Mathematics and Physics, Charles University and Institute of Computer Science, Academy of Sciences of the Czech Republic. E-mail: mirektuma@karlin.mff.cuni.cz*

SUMMARY

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 it to be effective. In this paper, the focus is on a class of incomplete factorization algorithms that can be used to compute preconditioners for symmetric indefinite systems. A limited memory approach is employed that incorporates a number of new ideas with the goal of improving the stability, robustness and efficiency of the preconditioner. These include the monitoring of stability as the factorization proceeds and the incorporation of pivot modifications when potential instability is observed. Numerical experiments involving test problems arising from a range of real-world applications demonstrate the effectiveness of our approach. Copyright © 2017 John Wiley & Sons, Ltd.

Received ...

KEY WORDS: sparse matrices, sparse linear systems, indefinite symmetric systems, iterative solvers, preconditioning, incomplete factorizations, pivoting.

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 (see Benzi et al. [1] for an overview). 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 general indefinite systems is the main focus of this paper.

A significant attraction of sparse direct methods for solving sparse indefinite systems is that they can often be used as black box solvers. Their main weakness is the amount of memory they require: for very large problems (typically those from three dimensional models) an iterative method must be used. Iterative methods may 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). To be effective iterative methods generally 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, significant 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

*Supported by EPSRC grant EP/I013067/1.
†Partially supported by the Grant Agency of the Czech Republic Projects GA13-06684S and 17-04150J.

in [1]. Other techniques that make use of the block structure include constraint preconditioners [2, 3], and symmetric-triangular preconditioners [4]. Alternatively, the saddle-point structure may be partially exploited. For example, the structure may be used as a 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 is available for symmetric quasi-definite (SQD) systems. Vanderbei [5] shows SQD matrices are strongly factorizable while a stability analysis is given by Gill et al. [6] (see also [7]).

An important contribution by Chow and Saad [8] considered the more general class of incomplete LU preconditioners, while the work of Li and Saad [9] represented an important step in the development of well-implemented general indefinite preconditioners. They integrated pivoting procedures [10, 11, 12] with scaling and reordering. Building on this, Greif, He, and Liu [13] recently developed a new incomplete factorization package SYM-ILDL.

In this paper, we consider incomplete factorizations of general indefinite systems; we generalize our previous work on incomplete factorizations for special classes of problems as well as proposing new ideas. In the positive-definite case, it was demonstrated in [14] that employing a modest amount of additional memory during the construction of the preconditioner can significantly improve its quality without increasing the number of entries in the incomplete factor and hence without increasing the preconditioner application cost. Moreover, this approach appears to outperform the modification scheme of Jennings and Malik [15, 16]. The implementation of our limited memory approach and the development of the corresponding HSL [17] software package are described in [18]. An extension of the technique to the computation of signed incomplete Cholesky factorizations for saddle-point problems was discussed in [19]. It was shown that our proposed approach allows a more general reordering of the rows and columns of the matrix than in segregated approaches that preserve the saddle-point structure, while employing additional memory in the construction of the preconditioner improves its quality and the use of two global shifts guards against breakdowns during the factorization.

In this paper, we consider the significantly harder general indefinite case. Our goal is to improve the stability, robustness and efficiency of incomplete factorization preconditioners. We incorporate and explore a number of ordering and pivoting strategies. Furthermore, we introduce diagonal modifications to improve stability and we generalize such modifications to the 2×2 pivots that are needed to maintain symmetry. Even with well-bounded entries in the incomplete factors, the triangular solves performed during each application of the preconditioner can be highly unstable. Thus another novel idea is the monitoring of stability as the factorization proceeds. If instability is detected, we can use a number of possible cures. First, the factorization can be restarted with a larger global shift (as in [14, 20]). Second, at essentially no additional time overhead in the construction of the preconditioner, the intermediate memory that was discarded in the positive-definite case after the factorization had completed can be employed to improve the quality of the preconditioner, albeit at the cost of a denser incomplete factor. We also propose a strategy based on an auxiliary optimization problem that allows us to improve the quality by using only some of the intermediate memory.

The rest of the paper is organised as follows. In Section 2, we briefly describe our incomplete factorization algorithm and the different pivoting strategies that it incorporates. 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 caused by the choice of pivots 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. In Section 6, our findings are summarised and some concluding comments are made.

DOI: 10.1002/nla

2. FACTORIZATION AND PIVOTING

2.1. *Limited-memory incomplete factorization*

We first summarize our limited-memory incomplete Cholesky (IC) factorization approach for a symmetric positive-definite A; this is implemented within the package `HSL_MI28` from the HSL mathematical software library [17, 18]. For such A, the computed IC factorization is of the form $(\Pi L)(\Pi L)^T$, where Π is a permutation matrix, chosen to preserve sparsity and L is lower triangular with positive diagonal entries. The matrix A is optionally scaled and, if necessary, shifted to avoid breakdown of the factorization (see Section 3). Thus the incomplete factorization of

$$\bar{A} = \bar{S}\Pi^T A \Pi \bar{S} + \alpha I$$

is computed, where $\bar{S}$ is a diagonal scaling matrix and α is a positive shift. The incomplete factorization preconditioner is $P = (\bar{L}\bar{L}^T)^{-1}$ with $\bar{L} = \Pi \bar{S}^{-1} L$.

The `HSL_MI28` algorithm is based on a limited memory version of the left-looking approach by Tismenetsky [21] and Kaporin [22]. The basic scheme employs a sparse matrix factorization of the form

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

Here R is a strictly lower triangular matrix with entries that are smaller in absolute value than those in L 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 [22], drop tolerances may be used to limit the memory required in the computation of the incomplete factorization. User-chosen parameters $lsize$ and $rsize$ are used to control the maximum number of fill entries in each column of L and the maximum number of entries in each column of R, respectively. The memory used for the preconditioner L can, to some extent, be replaced by the intermediate memory R (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. Further details and numerical results are given in [14, 18, 19].

To extend this approach, if A is symmetric indefinite, we replace (2.1) by

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

where L has unit diagonal entries, R is as before, D is block diagonal with 1×1 and 2×2 blocks, and E is of the form

$$E = RDR^T.$$

2.2. *Pivoting strategies*

For positive-definite problems, numerical pivoting is unnecessary and for saddle-point systems pivoting can be avoided by using an appropriately chosen constrained ordering [19]. In the general indefinite case, pivoting down the diagonal may be possible if the matrix is (close to) positive definite or if preprocessing of the matrix ensures large diagonal entries. However, in general, pivoting is needed for stability. The partial pivoting strategy of Bunch and Kaufman [11] has been widely used for factorizing (dense) symmetric indefinite matrices using 1×1 and 2×2 pivots (the latter are needed to maintain stability without destroying symmetry). The algorithm for choosing the i-th pivot may be outlined as follows.

The parameter α_p is chosen to minimize the global bound on growth of entries in the factors. 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) can adversely effect the sparsity of the computed factors. Consequently, it is common to use a

DOI: 10.1002/nla

Partial pivoting strategy 1: Bunch-Kaufman (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 1×1 pivot
else
    $\sigma := \max\{|a_{kj}|, k \neq j\}$
    if $|a_{ii}|\,\sigma \geq \alpha_p\,\lambda^2$ **then**
        use a_{ii} as a 1×1 pivot
    else if $|a_{jj}| \geq \alpha_p\,\sigma$ **then**
        use a_{jj} as a 1×1 pivot
    else
        use $\begin{pmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{pmatrix}$ as a 2×2 pivot
    end
end

threshold-based strategy that limits the search (see, for example, [23, 24]) but may compromise stability. Instead of choosing the largest off-diagonal entry in the first column of the reduced matrix to form the 2×2 pivot, Liu [25] proposed using a sparsity threshold $\tau \in (0, 1]$ and selecting the entry of smallest row index in the first column that satisfies a threshold condition. Liu's strategy is given below as partial pivoting strategy 2. The optimal α_p now depends on the value of τ. Liu shows that it satisfies a cubic equation and is monotonically increasing with respect to τ. In our codes we compute the optimal α_p directly solving the related cubic equation. Note that the value of α_p is then smaller than in the case of Bunch-Kaufman pivoting.

Partial pivoting strategy 2: Liu (1987) threshold pivoting

Choose a sparsity threshold value τ 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 1×1 pivot
else
    $\sigma := \max\{|a_{ks}|, k \neq s\}$
    if $|a_{ii}|\,\sigma \geq \alpha_p\,\lambda^2$ **then**
        use a_{ii} as a 1×1 pivot
    else if $|a_{ss}| \geq \alpha_p\,\sigma$ **then**
        use a_{ss} as a 1×1 pivot
    else
        use $\begin{pmatrix} a_{ii} & a_{si} \\ a_{si} & a_{ss} \end{pmatrix}$ as a 2×2 pivot
    end
end

For tridiagonal matrices it is possible to use a more localized pivoting strategy (partial pivoting strategy 3). For simplicity, we assume here that the subdiagonal entries are nonzero. This approach was used by Hagemann and Schenk [26] in combination with a symmetric version of a maximum weighted matching ordering for indefinite problems. The matching-based ordering is used to a priori symmetrically 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 [27, 28]). Hagemann and Schenk employ a local perturbation if the candidate pivot is too small to be inverted. Advantages of the permutation are that no entries beyond the diagonal and subdiagonal are searched and no

DOI: 10.1002/nla

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.

Partial pivoting strategy 3: Bunch tridiagonal pivoting (1974)
$\alpha_p := (\sqrt{5}-1)/2 \approx 0.62$
σ:= the entry of largest absolute value in the initial matrix
if $|a_{ii}|\,\sigma \geq \alpha_p\,|a_{i+1,i1}|^2$ **then**
use a_{ii} as a 1×1 pivot
else
use $\begin{pmatrix} a_{ii} & a_{i,i+1} \\ a_{i+1,i} & a_{i+1,i+1} \end{pmatrix}$ as a 2×2 pivot
end

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 [29] 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 popularize incomplete factorizations, but ad hoc local perturbations with no relation to the overall matrix can lead to large growth in the entries that is related to unstable preconditioners. Thus a more commonly used approach is the one originally suggested by Manteuffel [30] that involves factorizing the diagonally shifted matrix $A + \alpha I$ for some positive α. 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, for example, [14, 20]). In general, provided the problem has been well-scaled, α can be chosen to be small so that the shifted matrix is not far from the original one. Recent results [14, 18] illustrate the effectiveness of using a shift in increasing the stability of the factorization and by monitoring the diagonal entries as the factorization progresses, the extra work needed on restarting the factorization for each new shift is generally not prohibitive. Note that other approaches to modifying A originated in solving discretized partial differential equations. An example of a general sophisticated modification strategy is given in the paper by MacLachlan, Osei-Kuffuor and Saad [31]. However, they use a standard ILU factorization that does not employ an intermediate factor R but uses local modifications.

A relatively simple generalization of the global shift strategy was used by Scott and Tůma [19] for saddle-point systems. 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 [32]).

Shifts have also been used in the construction of incomplete factorizations of general sparse matrices. In particular, the package IFPACK [33] 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 real parameters that must be set by the user. The documentation for the code suggests a trial-and-error method for selecting suitable values. While our current interest is in

Prepared using nlaauth.cls
DOI: 10.1002/nla

real shifts, we observe that using complex shifts and switching to complex arithmetic is a possible alternative. This apparently works well in some particular applications (see, for example, [34, 35]).

For symmetric indefinite problems, we need a modification strategy that allows for both 1×1 and 2×2 pivots. Whilst it is always possible to choose a shift such that the diagonal blocks of the shifted matrix are sufficiently diagonally dominant for the factorization to be breakdown free, such a choice may lead to an inaccurate factorization of the unshifted matrix and hence to a poor quality preconditioner.

In the following, 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 strategy satisfies this assumption. Namely, the 2×2 pivot (i, j) is chosen only if both $|a_{jj}| < \alpha_p \sigma$ and $|a_{ii}|\sigma < \alpha_p \lambda^2$. Thus $|a_{ii}||a_{jj}| < \alpha_p^2 \lambda^2 < \lambda^2$.

In general, the stability of matrix factorizations is reflected in two quantities. The first is the *local growth factor* that measures possible growth in the magnitudes of the entries of the factors. It can be directly influenced by pivoting (that is, small pivots can lead to growth). Once a breakdown is encountered, the block diagonal of the matrix is globally modified. In this section, we show how, given a shift, the modifications can be done in an optimal way with respect to the growth of the other entries. The second quantity is the *instability factor* that expresses the fact that the triangular solves using the computed factors can be unstable. A large instability factor indicates stability problems and can occur even when the pivots are not particularly small; this is discussed in the next section.

Standard references on symmetric indefinite factorizations (including [12, 36]) derive formulae for the growth based on the *global* behavior of the pivoted factorization (see also [37]) and employ quantities such as the maximum magnitudes of the diagonal and off-diagonal entries of A. To examine the growth caused by shifting particular pivots, we introduce the idea of a *local growth factor*.

Definition 3.1
Consider a 1×1 or 2×2 (nonsingular) pivot P used in an indefinite factorization. The value θ of the entry of largest absolute value in P^{-1} is called the *local growth factor*.

If $P = p \in \mathcal{R} \setminus \{0\}$, the local growth factor is just $\theta = 1/|p|$. Consider now a 2×2 pivot P given by

$$P = \begin{pmatrix} a & b \\ b & c \end{pmatrix}, \qquad ac - b^2 \neq 0, \tag{3.2}$$

with inverse

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

This has local growth factor

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

We now consider how the local growth factor is influenced by a shift and, in particular, how θ can be decreased by the use of an appropriately chosen shift and/or multiplier. Throughout our discussion, $\alpha > 0$ and $\rho \geq 1$. For a 1×1 pivot, we make the modification

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

which reduces the local growth factor. 2×2 pivots must be considered more carefully. One possible approach to modifying a 2×2 pivot is to treat it as two consecutive 1×1 pivots. While the analysis of diagonal pivoting strategies starting with the seminal contribution of Bunch and Parlett [12] based on upper bounds of the magnitudes of entries, often uses this approach, local determination of the

DOI: 10.1002/nla

shift with possible restarts does not allow this. For example, for a 2×2 pivot with zeros on the diagonal, the first column does not update the second. Consequently, we discuss modifications for 2×2 pivots independently and separately from the motivation for 1×1 pivots. We follow [23] and distinguish three basic types of 2×2 pivots.

3.1. Oxo pivots

An *oxo* pivot is of the form

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

¿From (3.3), it has local growth factor

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

Thus stability of P_{oxo} is improved 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 nonzero 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 $\rho\, b + sgn(b)\, \alpha$. The local growth factor becomes

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

which is a non increasing function of α and ρ. Thus the local growth factor is either unchanged or is reduced, which is our goal. However, if $|a| > |\rho\, b + \mathrm{sgn}(b)\, \alpha|$, it can be reduced further by decreasing $|a|$. In addition, the Euclidean condition number decreases by decreasing $|a|$, as we see from the following result.

Lemma 3.1
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

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

DOI: 10.1002/nla

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. □

Lemma 3.1 and equation (3.4) imply a practical modification procedure: namely, modifying the off-diagonal entry to $\rho\, b + \mathrm{sgn}(b)\, \alpha$, reduces the local growth factor. Replacing a by $a - sgn(a)\, \delta$ with

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

further reduces the local growth factor. The modified tile pivot is

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

with 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 $b^2 > |ac|$ (Assumption 3.1) and, without loss of generality, we assume $|a| \geq |c|$. Again, stability is increased by modifying b to $\rho\, b + sgn(b)\alpha$. The local growth factor becomes

$$\frac{max(|\rho\, b + sgn(b)\, \alpha|, |a|)}{(|\rho\, b + sgn(b)\, \alpha)^2},$$

which is a non increasing function of α and ρ. As in the case of a tile pivot, the local growth factor can be reduced further by decreasing $|a|$. To show this, we employ the following result.

Lemma 3.2
Assume $b^2 - |ac| > 0$, $|a| > |c| > 0$. Then

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

is a decreasing function of $|a|$.

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

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

By assumption, $b^2 - |ac| > 0$ so that $c^2 + 2b^2 - ac$ is positive. Since $a + c > 0$, it follows that the derivative is negative and q is a decreasing function of $a > 0$. If a is negative, the derivative of q is positive and by decreasing the entry $-a > 0$ we obtain the same conclusion. We conclude that q is a decreasing function of $|a|$. □

DOI: 10.1002/nla

Lemma 3.3
The Euclidean condition number of P_{full} under the assumptions from Lemma 3.2 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

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

As in Lemma 3.1, this can be rewritten as

$$1 + 1/2q + \left(\sqrt{1/q^2+4/q}\right)/2,$$

and the result follows from Lemma 3.2. □

Lemma 3.3 implies that for $|a| > |\rho\, b + sgn(b)\, \alpha|$, we can simultaneously decrease the local growth factor and the condition number of P_{full}. Note that if $a = -c$, the condition number is equal to 1. In practice, we again limit the size of the modification to a by using $a - sgn(a)\, \delta$ with δ given by (3.5). The modified full 2×2 pivot is

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

with local growth factor

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

4. LOCAL INSTABILITY

A well-studied problem in sparse symmetric indefinite factorizations as well as in sparse non symmetric factorizations is the growth in the size of the entries of the factors. 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 [8], 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

$$condest = condest(L) = ||(LDL^T)^{-1}e||_\infty, \tag{4.1}$$

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 [33] also uses $condest$ and, as already discussed, employs *a priori* diagonal perturbations if the condition estimate is larger than machine precision.

In this section, we propose monitoring for possible instability as the factorization proceeds. If instability is detected, the factorization may be restarted with a new shift or new multiplier or with different parameter settings, enabling us to obtain more robust incomplete symmetric indefinite factorizations that provide higher quality preconditioners. Monitoring stability may 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.

DOI: 10.1002/nla

The factors L, D and R are computed in ν steps, where ν is equal to n minus the number of 2×2 pivots. For $k = 1, \ldots, \nu$, let L_k, R_k and D_k denote the leading principal submatrices of order n_k (with $n_\nu \equiv n$) of L, R and D, respectively. Further, let p_k be the size of the k-th pivot ($p_k = 1$ or 2). To monitor stability as the factorization proceeds, we need a computable quantity that can be cheaply updated throughout the factorization. We will call it the *instability factor* and define it as follows.

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

Consider the factor L_k expressed in the bordered form

$$L_1 = I_1, \qquad L_k = \begin{pmatrix} L_{k-1} & \\ l_k & I_k \end{pmatrix}, \qquad k = 2, \ldots, \nu,$$

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

Computation of the instability factor g_k

$v_1 = e_1$
$g_1 = 1$
for $k = 2, \ldots, \nu$

$$v_k = |L_k^{-1}|e_k = \left| \begin{pmatrix} L_{k-1} & \\ l_k & I_k \end{pmatrix}^{-1} \right| e_k \equiv \left| \begin{pmatrix} L_{k-1}^{-1} & \\ -l_k L_{k-1}^{-1} & I_k \end{pmatrix} \right| e_k \equiv \begin{pmatrix} v_{k-1} \\ |l_k v_{k-1}| + e_{p_k} \end{pmatrix}$$

$$g_k = max(g_{k-1}, ||(v_k)_{n_{k-1}+1:n_k}||_\infty)$$

end

This computation requires us to store the previous instability factor g_{k-1} and a vector v_k of length n_k. Entries $n_{k-1} + 1 : n_k$ of v_k are computed using

$$(v_k)_{n_{k-1}+1:n_k} = e_{p_k} + |l_k v_{k-1}|. \tag{4.2}$$

The following lemma shows that computation of the instability factor leads to an incrementally computable upper bound for $condest$.

Lemma 4.1
Assume that the diagonal entries of the computed LDL^T factorization satisfy $||D^{-1}||_\infty \leq \beta$. Then the instability factor g and $condest$ are related by the inequality

$$condest \leq \beta g^2.$$

Proof
¿From (4.1) it follows that $condest \equiv ||L^{-T}D^{-1}L^{-1}e||_\infty$ can be bounded as follows

$$||L^{-T}D^{-1}L^{-1}e||_\infty \leq ||L^{-T}D^{-1}L^{-1}e||_1 = e^T|L^{-T}D^{-1}L^{-1}e| \leq e^T|L^{-T}||D^{-1}||L^{-1}|e \leq \beta g^2.$$

□

Note that the shift strategies for 1×1 and 2×2 pivots discussed in Section 3 are needed to bound D^{-1}. We have observed that the factorization appears stable for $g \ll \sqrt{1/\epsilon}$ (where ϵ is the machine precision) and for "hard" indefinite systems, adding the computed R factor to L can reduce g for the resulting L. Thus if g is large, $L + R$ may give a more efficient preconditioner, albeit one that requires more memory than L alone. Alternatively, entries of R that lead to a decrease in the size of

DOI: 10.1002/nla

g can be selectively added to L; this is illustrated in Section 5. Thus R can be regarded as a source of additional entries that can potentially improve the quality of the final preconditioner. We propose a 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.

Let the factor R_k be written in the bordered form

$$R_1 = 0_1, \qquad R_k = \begin{pmatrix} R_{k-1} & \\ r_k & 0_k \end{pmatrix}, \qquad k = 2, \ldots, \nu,$$

where 0_k is the null matrix of dimension p_k and r_k is the $p_k \times n_{k-1}$ block of the off diagonal entries in the k-th (block) row of R. Denote by λ_k the set of column indices $j \in \{1, \ldots, n_{k-1}\}$ for which $(l_k)_{1:p_k,j}$ is nonzero. Equation (4.2) that is used to compute g can then be written as

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

But our goal is to minimize the sum based on L^{-1} rather than $|L^{-1}|$. In particular, we would like to find a subset γ_k of column indices $j \in \{1, \ldots, n_{k-1}\}$ for which $(r_k)_{1:p_k,j}$ is nonzero and which minimizes the sum

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

Let $\bar{l}_k$ (respectively, $\bar{r}_k$) be the $p_k \times n_{k-1}$ block with entries $(\bar{l}_k)_{ij} = (l_k)_{ij}(v_{k-1})_j$ (respectively, $(\bar{r}_k)_{ij} = (r_k)_{ij}(v_{k-1})_j$), $i = 1 : p_k$, $j = 1, \ldots, n_{k-1}$. The minimization problem is then defined as follows.

Problem 4.1
For $k = 2, \ldots, n_\nu$, find a subset γ_k of the column indices $j \in \{1, \ldots, n_{k-1}\}$ that minimizes over all such choices some norm of

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

Finding an approximate solution of this problem is a special instance of the sparse approximation problem of selecting a small number of columns in a source matrix such that their sum approximates a target matrix. It is a matrix generalization of the *subset sum selection problem* [38] in which the matrix has at most two rows. Here we propose a simple greedy strategy to select the subset γ_k for Problem 4.1. The minimization is based on evaluating 1-norms of column blocks $\bar{l}_k$ and $\bar{r}_k$, that is, on their absolute values or on the sum of the absolute values in the case of a 2×2 block ($p_k = 2$). Our algorithm is below. At each stage, sum^+ is the current value of the objective function in Problem 4.1; the output is a set of column indices γ_k that gives an approximate solution to Problem 4.1. Entries of R that correspond to γ_k for $k = 2, \ldots, n_\nu$ form a factor Rs. Our experiments illustrate that, if $condest(L)$ is large, then if L is replaced by $L + Rs$, $condest(L + Rs)$ is typically smaller, giving a higher quality preconditioner.

DOI: 10.1002/nla

Simple greedy strategy for finding a subset of column indices γ_k **(**$k = 2, \ldots, n_\nu$**)**

Let p_k be the size of the k-th pivot ($p_k = 1$ or 2)
Let λ_k be the set of column indices j for which $(\bar{l}_k)_{1:p_k,j}$ is nonzero
Let $\bar{\gamma}_k$ be the set of column indices j for which $(\bar{r}_k)_{1:p_k,j}$ is nonzero
Set $\gamma_k = \emptyset$
Set $sum = e_{p_k} - \sum_{j \in \lambda_k} (\bar{l}_k)_{1:p_k,j}$
do while $\bar{\gamma}_k \neq \emptyset$
    $j_\rho = \text{argmin}_{j \in \bar{\gamma}_k} ||sum - (\bar{r}_k)_{1:p_k,j}||_1$
    $sum^+ = sum - (\bar{r}_k)_{1:p_k,j_\rho}$
    if $||sum^+||_1 < ||sum||_1$ **then**
        $\bar{\gamma}_k = \bar{\gamma}_k \setminus \{j_\rho\}$
        $\gamma_k = \gamma_k \cup \{j_\rho\}$
        $sum = sum^+$
    else
        exit
    end
end do

5. NUMERICAL EXPERIMENTS

5.1. Test environment

Throughout this section, we refer to the implementation of our incomplete factorization algorithm with pivoting as `PISIF` (Pivoted Incomplete Symmetric Indefinite Factorization). All the software we have developed to obtain the experimental results is written in Fortran and the gfortran Fortran compiler (version 4.8.2) with option -O3 is used. All reported times are elapsed times in seconds measured using the Fortran `system_clock` and are for running on a machine with two Intel Xeon E5620 quadcore processors and 24 Gbytes of memory. Sparse matrix-vector products are performed using the Intel MKL routine `mkl_dcsrsymv`. The implementation of the GMRES(1000) algorithm (with right preconditioning) [39] 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 $x = e$, and stopping criteria

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

where $\hat{x}$ is the computed solution. In addition, for each test we impose a limit of 2000 iterations. Following [40], in our experiments we define the *efficiency* of the preconditioner $P = (LDL^T)^{-1}$ to be

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

where $iter$ is the iteration count. The lower the value of (5.1), the better the preconditioner. *efficiency* reflects the number of floating-point operations performed using the preconditioner during the iterative method but as it does not include communication, it may not reflect the runtime of the iterative method and so we also use the runtime of our prototype code in some of our experiments to assess performance. We also define the fill ratio in the incomplete factor to be

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

For comparison purposes, we use the multicore sparse direct solver `HSL_MA97` [41]. Preordering of A is performed using the HSL packages `HSL_MC68` and `HSL_MC80`. Our test problems are real indefinite matrices taken from the University of Florida (UFL) Sparse Matrix Collection [42].

DOI: 10.1002/nla

5.2. *Results for interior-point optimization matrices*

Test Set 1 are interior-point optimization matrices that are of saddle-point form, namely,

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

with H $n \times n$ symmetric positive definite, B rectangular $m \times n$ ($m \leq n$), and $C = 10^{-8} I_m$ (where I_m is the $m \times m$ identity matrix). In Table I, we report the fill ratio $fill_L$ and the run times for the direct solver `HSL_MA97` using `MC64` scaling [43, 44], which has been found to work well when solving "tough" indefinite systems [28, 45]; the default (nested dissection) ordering is used. The interior-point examples we have selected all have $fill_L > 17.0$. For a direct solver, the amount of fill is highly dependent on the choice of ordering; we experimented with other orderings, including approximate minimum degree (AMD) and matching-based orderings, but for these interior-point examples the solution times as well as $fill_L$ were significantly greater.

Table I. 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, $fill_L$ is the ratio of the number of entries in the complete factor of A for the default ordering to $nz(A)$ and Time is the solution time (in seconds) for the direct solver `HSL_MA97`.

Identifier	n	m	$nz(A)$	$fill_L$	Time
GHS_indef/c-55	19121	13659	218115	21.5	0.49
GHS_indef/c-59	23813	17469	260909	17.6	0.59
Schenk_IBMNA/c-62	25158	16573	300536	28.5	1.07
GHS_indef/c-68	36546	28264	315403	29.2	1.48
GHS_indef/c-71	44814	31824	468079	37.1	2.60
Schenk_IBMNA/c-big	201877	143364	1343050	39.0	10.3

Table II. The performance of `PISIF` preconditioned GMRES(1000) on Test Set 1. None denotes no preordering, AMD denotes approximate minimum degree ordering and match(AMD) denotes AMD combined with matching. $iter$ is the number of GMRES iterations, and T_f, T_g and T denote, respectively, the times (in seconds) to compute the incomplete factorization (including preordering and scaling the matrix), to run GMRES and the total solution time. – denotes failure to converge within 2000 iterations.

Identifier	None				AMD				match(AMD)			
	$iter$	T_f	T_g	T	$iter$	T_f	T_g	T	$iter$	T_f	T_g	T
GHS_indef/c-55	1956	2.84	24.4	27.2	78	2.90	0.24	3.15	30	0.54	0.08	0.61
GHS_indef/c-59	727	8.01	8.29	16.3	41	5.97	0.13	6.10	28	0.50	0.09	0.59
Schenk_IBMNA/c-62	–	–	–	–	426	4.73	3.90	8.62	26	0.62	0.12	0.74
GHS_indef/c-68	711	18.31	11.7	30.0	102	10.3	0.62	10.9	6	1.59	0.03	1.62
GHS_indef/c-71	655	37.71	12.9	50.6	152	23.5	1.47	25.0	21	1.51	0.13	1.64
Schenk_IBMNA/c-big	–	–	–	–	13	91.3	0.41	91.7	9	8.54	0.27	8.81

For `PISIF`, if $lsize$ and $rsize$ are held constant, the amount of fill in the incomplete factors is essentially independent of the ordering of A that is used. However, the ordering can effect the quality of the preconditioner. The results in Table II are for `PISIF` with $lsize = rsize = 10$, Bunch Kaufman pivoting and `MC64` scaling, and compare no preordering with approximate minimum degree (AMD) and with a matching-based ordering combined with AMD (that is, match(AMD), which applies AMD to the compressed graph that is employed within the matching algorithm; see [28] for details). For each example $fill_{IL} < 3.0$, illustrating the incomplete factors are significantly sparser than the complete factors. We see that using a matching-based ordering leads to a much faster time for computing the incomplete factorization and to higher quality preconditioners. In general,

DOI: 10.1002/nla

having performed the matching ordering, pivoting is not needed (that is, the Bunch Kaufman algorithm makes few modifications to the supplied pivot order), and this accounts for the significant reduction in the factorize time and can result in the iterative solver outperforming the direct solver. We remark that, although we omit the full details here, we investigated using other orderings (such as a nested dissection ordering in place of AMD). We found that, for these test problems, match(AMD) was generally the best choice in terms of the preconditioner quality.

We next provide a comparison with the signed incomplete Cholesky factorization approach of [19] (HSL_MI30) and with SYM-ILDL [13]. The PISIF and HSL_MI30 results in Table III use $lsize = rsize = 30$. For PISIF, we again use match(AMD) ordering, Bunch Kaufman pivoting and MC64 scaling. Based on findings in [19], the settings for HSL_MI30 are initial shifts $\alpha_{in}(1:2) = 0.01$, Sloan profile-reducing ordering, and equilibration scaling. For SYM-ILDL, we use equilibration scaling, AMD ordering and the parameter settings $fill = 12.0$ and $tol = 0.003$). These choices for SYM-ILDL give a similar amount of fill as for PISIF and HSL_MI30. The results show that PISIF generally performs strongly (in terms of fill, iteration count and efficiency) and illustrates that a general-purpose indefinite algorithm can outperform one that exploits the saddle-point structure.

Table III. GMRES(1000) convergence results using PISIF and HSL_MI30 preconditioning applied to Test Set 1 ($lsize = rsize = 30$). Results are also given for SYM-ILDL. – denotes failure to converge within 2000 iterations.

Identifier	PISIF			HSL_MI30			SYM-ILDL		
	$fill_{IL}$	*efficiency*	*iter*	$fill_{IL}$	*efficiency*	*iter*	$fill_{IL}$	*efficiency*	*iter*
GHS_indef/c-55	3.73	9.0×10^{6}	11	3.37	3.0×10^{7}	41	4.35	7.4×10^{7}	78
GHS_indef/c-59	4.04	1.3×10^{7}	12	3.67	3.9×10^{7}	41	4.33	1.1×10^{8}	97
Schenk_IBMNA/c-62	3.45	1.2×10^{7}	12	3.40	6.2×10^{7}	61	3.23	3.3×10^{7}	34
GHS_indef/c-68	3.48	3.3×10^{6}	3	4.06	1.8×10^{7}	14	4.31	8.3×10^{7}	61
GHS_indef/c-71	3.94	1.8×10^{7}	10	3.51	8.7×10^{7}	53	4.03	7.0×10^{7}	37
Schenk_IBMNA/c-big	3.08	2.1×10^{7}	5	4.44	3.3×10^{8}	56	4.47	–	–

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 $m \times m$ null matrix. Note that, as H is not positive definite, a signed incomplete Cholesky factorization is not recommended. The problems are listed in Table IV. HSL_MA97 is run with match(AMD) ordering and MC64 scaling. Note that, although of full structural rank, these problems are not all of full numerical rank. We further observe that although the direct solver works well for these examples (with $fill_L < 6.5$), they are included in this study since they demonstrate the potential benefits of using a nonzero shift and multiplier, and of using $L + R$ as the preconditioner.

Table IV. TSOPF test problems (Test Set 2).

Identifier	n	m	$nz(A)$	HSL_MA97	
				$fill_L$	Time
TSOPF_FS_b39_c7	14118	14098	368599	6.25	0.14
TSOPF_FS_b39_c19	38118	38098	998359	6.32	0.37
TSOPF_FS_b39_c30	60118	60098	1575639	6.16	0.50
TSOPF_FS_b162_c3	15424	15374	904612	6.38	0.38
TSOPF_FS_b162_c4	20424	20374	1204322	6.38	0.52

DOI: 10.1002/nla

In Table V, results are given for PISIF with $lsize = rsize = 30$ using match(AMD) ordering, MC64 scaling, and Bunch tridiagonal pivoting. The first two sets of three columns use L as the preconditioner while the last two sets of three columns use $L + R$. We present results for shifts $\alpha = 0$ and 0.01. With $\alpha = 0$, for problems TSOPF_FS_b39_c7 and TSOPF_FS_b39_c19 the factorization suffers breakdown (that is, at some stage, a stable pivot cannot be found and we terminate the computation). For Test Set 2, we found that using $\alpha < 0.01$ generally led to a poorer quality preconditioner and using a nonzero 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.

Table V. Results for PISIF with $lsize = rsize = 30$, with and without a nonzero 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_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_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

Table VI. Timings for PISIF with $lsize = rsize = 30$ and $alpha = 0.01$, using L and $L + R$ as the preconditioner. T_f, T_g and T denote, respectively, the times (in seconds) to compute the incomplete factorization (including preordering and scaling the matrix), to run GMRES and total solution time.

Identifier	L			$L + R$		
	T_f	T_g	T	T_f	T_g	T
TSOPF_FS_b39_c7	0.26	1.66	1.92	0.28	1.04	1.33
TSOPF_FS_b39_c19	0.76	9.03	9.80	0.88	4.15	5.03
TSOPF_FS_b39_c30	1.68	30.8	32.4	1.96	12.2	14.1
TSOPF_FS_b162_c3	0.81	0.87	1.68	0.90	0.68	1.57
TSOPF_FS_b162_c4	1.12	1.35	2.47	1.23	1.04	2.27

Table VII. Results for PISIF with $lsize = rsize = 30$, with and without non-unit multiplier ρ, using L as the preconditioner. BD denotes factorization breakdown. – indicates convergence not achieved.

Identifier	$\rho = 1.0$				$\rho = 1.1$			
	$fill_{IL}$	*efficiency*	iters	*condest*	$fill_{IL}$	*efficiency*	iters	*condest*
TSOPF_FS_b39_c7	BD	BD	BD	BD	2.58	9.2×10^{8}	965	5.06×10^{8}
TSOPF_FS_b39_c19	BD	BD	BD	BD	2.59	–	–	3.12×10^{8}
TSOPF_FS_b39_c30	2.30	–	–	2.33×10^{20}	2.20	–	–	4.15×10^{9}
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}
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}

Timings are reported in Table VI. The increase in time for constructing the $L + R$ preconditioner comes from summing the computed L and R. This additional cost is more than offset by the reduction in the GMRES time but the total time is greater than the direct solver time.

DOI: 10.1002/nla

So far, we have used a fixed global multiplier $\rho = 1.0$. In Table VII, results are given for $\rho = 1.1$ with $\alpha = 0.0$ and L 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 but it is less effective than using a positive shift.

5.4. Results for density functional theory matrices

We now consider symmetric indefinite problems that come 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 VIII; these problems belong to 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 on our test machine because of insufficient memory.

Table VIII. PARSEC test problems (Test Set 3). – denotes the factorization failed because of 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 for GMRES(1000) with `PISIF` preconditioning are given in Tables IX and X; results are for both L and $L + R$ used as the preconditioner. In these experiments, we employ Bunch Kaufman pivoting, Sloan ordering and `MC64` scaling. In previous experiments, we employed a fixed shift ($\alpha = 0.0$ or 0.01); for these harder problems, we monitor the instability factor and if at any stage it exceeds 10^8, we increase the shift (starting with an initial value of 0.0) and restart the factorization. This process may need to be repeated more than once; the final α is given in column 2 of Table IX and the times in Table X include any time taken by restarting the factorization. For problems that used $\alpha > 0.0$ we found that, if we did not allow an increase in the shift, $condest$ is large and there is no convergence. For problems Si34H36 and Si87H76, $condest(L)$ remains large, but $condest(L + R)$ is significantly smaller and using $L + R$ improves the preconditioner quality and gives convergence. In all cases, $L + R$ reduces the iteration count and this reduction is sufficient to reduce the total computational time (although *efficiency* is smaller from some problems when L is used). As observed 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 $L + R$.

Comparing the timings reported in Table X with those for the direct solver given in Table VIII, 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 the direct solver `HSL_MA97` (although when it is successful, `HSL_MA97` again computes a solution of higher accuracy). For the PARSEC test problems, we

DOI: 10.1002/nla

Table IX. Results for PISIF with $lsize = rsize = 10$, with and without a nonzero shift, using L and $L + R$ as the preconditioner. – indicates convergence not achieved within 2000 iterations.

Identifier	α	L				$L+R$			
		$fill_{IL}$	*efficiency*	iters	*condest*	$fill_{IL}$	*efficiency*	iters	*condest*
CO	0.0	1.54	4.1×10^{8}	68	2.43×10^{2}	2.11	4.6×10^{8}	56	2.08×10^{1}
Ga10As10H30	0.0	1.24	2.7×10^{9}	688	5.22×10^{6}	1.61	1.3×10^{9}	255	2.49×10^{2}
Ga19As19H42	0.01	1.11	5.3×10^{9}	1068	1.39×10^{8}	1.40	2.5×10^{9}	390	2.35×10^{4}
Ga3As3H12	0.0	0.85	2.1×10^{8}	81	2.19×10^{1}	1.05	2.4×10^{8}	75	2.30×10^{1}
Ga41As41H72	0.01	1.11	7.3×10^{9}	704	4.35×10^{3}	1.39	7.3×10^{9}	557	1.13×10^{2}
GaAsH6	0.0	1.14	1.1×10^{8}	56	6.67×10^{1}	1.50	1.3×10^{8}	52	8.25×10^{1}
Ge87H76	0.02	1.00	2.3×10^{9}	563	1.61×10^{2}	1.28	2.4×10^{9}	464	2.54×10^{1}
Ge99H100	0.02	0.98	2.6×10^{9}	613	2.02×10^{2}	1.24	2.5×10^{9}	476	2.68×10^{1}
H2O	0.0	1.56	5.3×10^{7}	30	4.94	2.15	6.6×10^{7}	27	8.59
Si10H16	0.0	1.23	9.1×10^{7}	166	5.21×10^{5}	1.62	8.3×10^{7}	115	4.27×10^{2}
Si34H36	0.0	1.22	–	–	9.64×10^{11}	1.59	1.8×10^{9}	439	2.84×10^{4}
Si41Ge41H72	0.01	0.91	4.0×10^{9}	574	3.50×10^{3}	1.15	4.2×10^{9}	483	1.34×10^{2}
Si5H12	0.0	1.46	2.2×10^{7}	40	3.24×10^{1}	1.98	2.5×10^{7}	33	3.28×10^{1}
Si87H76	0.04	1.33	–	–	3.29×10^{12}	1.77	9.5×10^{9}	984	4.73×10^{5}
SiO	0.0	1.37	4.0×10^{7}	43	7.39×10^{1}	1.87	4.4×10^{7}	35	6.91×10^{1}
SiO2	0.0	0.77	1.9×10^{8}	43	4.17×10^{1}	1.04	2.3×10^{8}	39	4.83×10^{1}

Table X. Times for the incomplete factorization preconditioner PISIF computed using $lsize = rsize = 10$; L and $L + R$ are reported on. T_f, T_g and T denote, respectively, the time (in seconds) to compute the incomplete factorization (including preordering and scaling the matrix), to run GMRES and total solution time. For each problem, the lowest total time (and any within 10% of that time) is in bold. – denotes convergence not achieved within 2000 iterations.

Identifier	L			$(L+R)$		
	T_f	T_g	T	T_f	T_g	T
CO	2.79	1.99	**4.77**	3.03	2.03	**5.06**
Ga10As10H30	2.07	19.9	22.0	2.16	6.74	**8.91**
Ga19As19H42	4.77	47.0	51.8	5.14	15.8	**20.9**
Ga3As3H12	1.70	1.04	**2.74**	1.65	1.11	**2.76**
Ga41As41H72	9.03	77.8	86.8	9.32	59.4	**68.7**
GaAsH6	1.06	0.54	**1.60**	1.16	0.59	**1.75**
Ge87H76	6.70	21.5	28.2	6.28	17.0	**23.3**
Ge99H100	7.36	25.1	32.5	7.63	18.0	**25.6**
H2O	0.81	0.23	**1.04**	0.84	0.29	**1.12**
Si10H16	0.24	0.45	**0.69**	0.28	0.36	**0.63**
Si34H36	–	–	–	1.82	13.2	**15.0**
Si41Ge41H72	7.95	38.6	46.5	8.33	31.1	**39.4**
Si5H12	0.25	0.09	**0.34**	0.24	0.11	**0.35**
Si87H76	–	–	–	7.97	120	**128**
SiO	0.40	0.18	**0.58**	0.41	0.19	**0.61**
SiO2	2.62	1.05	**3.67**	2.79	1.13	**3.92**

thus have a potentially attractive alternative to a sparse direct solver that requires substantially less memory and computational time.

5.5. Monitoring instability and enriching L by entries from R

The following experiments further demonstrate the relationship between preconditioning using L, $L + R$ and also $L + Rs$ and they show the monitoring role of the instability factor. To separate the roles of shifts and the instability factor in our experiments, the shift is increased only if the factorization breaks down. Moreover, we will use *condest* to explore the usefulness of employing $L + R$ instead of L. Note that repeatedly increasing the shift α will eventually make *condest* small,

DOI: 10.1002/nla

but then the factorized matrix $A + \alpha I$ may be far from the original A. In this case, enhancing L using entries R can be beneficial and the instability factor may be a tool to decide when to do this, as we discuss below.

We first consider problem PARSEC/Ga3As3H12; we use the same scaling, ordering and pivot strategy for `PISIF` as in Section 5.4 and set shift $\alpha = 0.01$ and multiplier $\rho = 1.0$. 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

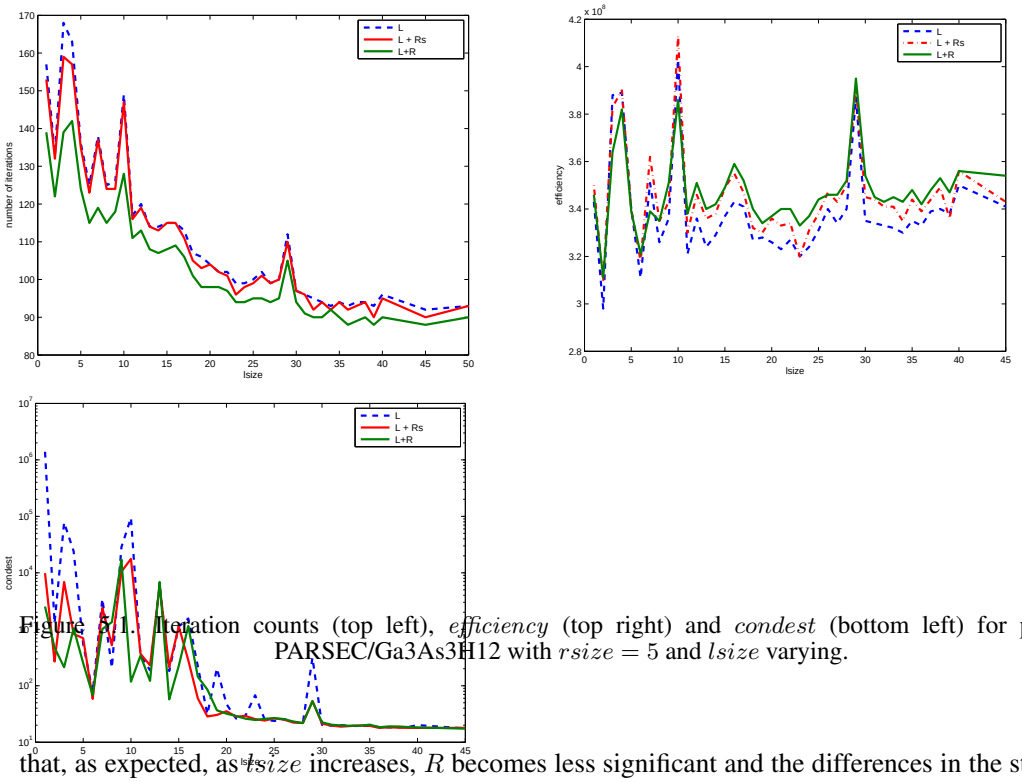

Figure 5.1. Iteration counts (top left), *efficiency* (top right) and *condest* (bottom left) for problem PARSEC/Ga3As3H12 with $rsize = 5$ and $lsize$ varying.

that, as expected, as $lsize$ increases, R becomes less significant and the differences in the statistics for L, $L + Rs$ and $L + R$ reduce. Moreover, the prediction of possible instability 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 that 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 the factor size.

We next consider problem PARSEC/Si10H16. In Table XI, 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 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 factor indicates there is no instability, the sparser L can be used without including entries of R.

DOI: 10.1002/nla

Table XI. Iteration counts and *condest* for problem PARSEC/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}

Figure 5.2 reports iteration counts and the *efficiency* for problem Schenk_IBMNA/c-56. For this example, as $lsize = rsize$ is increased, *condest* slowly decreases from approximately 10^6 to 10^4 and this is reflected in the value of the computed instability factor estimate g. If the value of g is modest, we can 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 minimize the iteration count. For this problem, using $L + Rs$ does not offer advantages over using L.

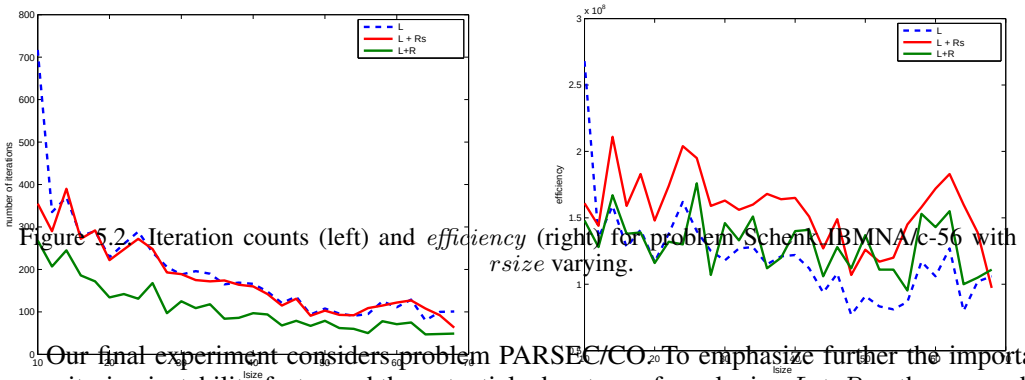

Figure 5.2. Iteration counts (left) and *efficiency* (right) for problem Schenk_IBMNA/c-56 with $lsize = rsize$ varying.

Our final experiment considers problem PARSEC/CO. To emphasize further the importance of monitoring instability factor and the potential advantage of employing $L + R$ as the preconditioner, we use the Liu partial pivoting strategy with a threshold of 0.1. This leads to faster computation of the incomplete factorization compared to using Bunch Kaufman pivoting but it can result in an unstable factorization. Iteration counts and *efficiency* are reported in Figure 5.3 for $lsize = rsize$ in the range 10 to 68. We see that using $L + R$ as the preconditioner stabilizes the Liu threshold 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 used as the preconditioner.

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 improved robustness of IC preconditioners 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.

DOI: 10.1002/nla

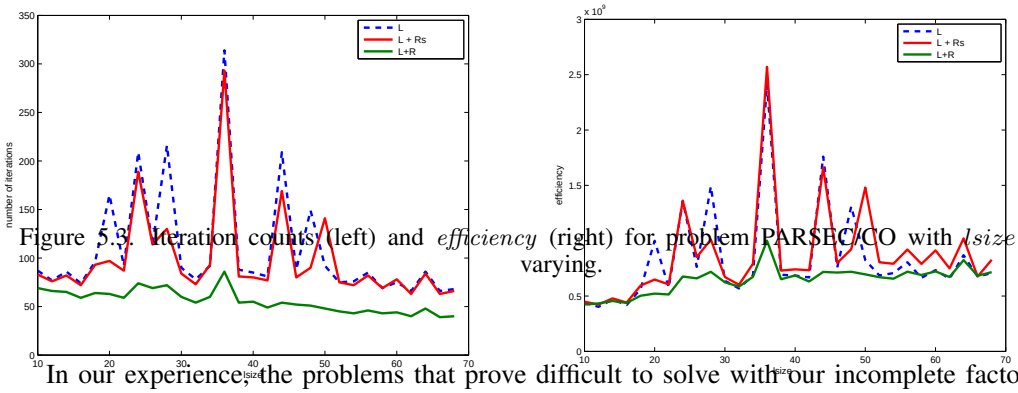

Figure 5.3. Iteration counts (left) and *efficiency* (right) for problem PARSEC/CO with $lsize = rsize$ varying.

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.

ACKNOWLEDGEMENTS

We would like to thank two anonymous reviewers for their constructive feedback.

REFERENCES

1. Benzi M, Golub G, Liesen J. Numerical solution of saddle point problems. *Acta Numerica* 2005; **14**:1–137.
2. Keller C, Gould NIM, Wathen AJ. Constraint preconditioning for indefinite linear systems. *SIAM Journal on Matrix Analysis and Applications* 2000; **21**(4):1300–1317.
3. Lukšan L, Vlček J. Indefinitely preconditioned inexact Newton method for large sparse equality constrained non-linear programming problems. *Numerical Linear Algebra with Applications* 1998; **5**(3):219–247.
4. Wu X, Golub GH, Cuminato JA, Yuan JY. Symmetric-triangular decomposition and its applications Part II: Preconditioners for indefinite systems. *BIT* 2008; **48**(1):139–162.
5. Vanderbei RJ. Symmetric quasidefinite matrices. *SIAM Journal on Optimization* 1995; **5**(1):100–113.
6. Gill PE, Saunders MA, Shinnerl JR. On the stability of Cholesky factorization for symmetric quasidefinite systems. *SIAM Journal on Matrix Analysis and Applications* 1996; **17**(1):35–46.

DOI: 10.1002/nla

7. Golub GH, Van Loan CF. Unsymmetric positive definite linear systems. *Linear Algebra and its Applications* 1979; **28**:85–97.
8. Chow E, Saad Y. Experimental study of *ILU* preconditioners for indefinite matrices. *Journal of Computational and Applied Mathematics* 1997; **86**(2):387–414.
9. Li N, Saad Y. Crout versions of ILU factorization with pivoting for sparse symmetric matrices. *Electronic Transactions on Numerical Analysis* 2005; **20**:75–85.
10. Ashcraft C, Grimes RG, Lewis JG. Accurate symmetric indefinite linear equation solvers. *SIAM Journal on Matrix Analysis and Applications* 1999; **20**(2):513–561.
11. Bunch JR, Kaufman L. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation* 1977; **31**:162–179.
12. Bunch JR, Parlett B. Direct methods for solving symmetric indefinite systems of linear systems. *SIAM Journal on Numerical Analysis* 1971; **8**:639–655.
13. Greif C, He S, Liu P. SYM-ILDL: C++ package for incomplete factorizations of symmetric indefinite matrices 2013. https://github.com/inutard/matrix-factor.
14. Scott JA, Tůma M. On positive semidefinite modification schemes for incomplete Cholesky factorization. *SIAM Journal on Scientific Computing* 2014; **36**(2):A609–A633.
15. Jennings A, Malik GM. Partial elimination. *Journal of the Institute of Mathematics and its Applications* 1977; **20**(3):307–316.
16. Jennings A, Malik GM. The solution of sparse linear equations by the conjugate gradient method. *International Journal of Numerical Methods in Engineering* 1978; **12**(1):141–158.
17. HSL. A collection of Fortran codes for large-scale scientific computation 2016. http://www.hsl.rl.ac.uk.
18. Scott JA, Tůma M. HSL_MI28: an efficient and robust limited-memory incomplete Cholesky factorization code. *ACM Transactions on Mathematical Software* 2014; **40**(4):Art. 24, 19.
19. Scott JA, Tůma M. On signed incomplete Cholesky factorization preconditioners for saddle-point systems. *SIAM Journal on Scientific Computing* 2014; **36**(6):A2984–A3010.
20. Lin CJ, Moré JJ. Incomplete Cholesky factorizations with limited memory. *SIAM Journal on Scientific Computing* 1999; **21**(1):24–45.
21. Tismenetsky M. A new preconditioning technique for solving large sparse linear systems. *Linear Algebra and its Applications* 1991; **154–156**:331–353.
22. Kaporin IE. 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* 1998; **5**:483–509.
23. Duff IS, Gould NIM, Reid JK, Scott JA, Turner K. Factorization of sparse symmetric indefinite matrices. *IMA Journal of Numerical Analysis* 1991; **11**:181–204.
24. Reid JK, Scott JA. Partial factorization of a dense symmetric indefinite matrix. *ACM Transactions on Mathematical Software* 2011; **38**. Article 10, 19 pages.
25. Liu JWH. A partial pivoting strategy for sparse symmetric matrix decomposition. *ACM Transactions on Mathematical Software* 1987; **13**(2):173–182.
26. Hagemann M, Schenk O. Weighted matchings for preconditioning symmetric indefinite linear systems. *SIAM Journal on Scientific Computing* 2006; **28**(2):403–420.
27. Duff I, Pralet S. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM Journal on Matrix Analysis and Applications* 2005; **27**:313 – 340.
28. Hogg JD, Scott JA. Pivoting strategies for tough sparse indefinite systems. *ACM Transactions on Mathematical Software* 2013; **40**. Article 4, 19 pages.
29. Kershaw DS. The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations. *Journal of Computational Physics* 1978; **26**:43–65.
30. Manteuffel TA. An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation* 1980; **34**:473–497.
31. MacLachlan S, Osei-Kuffuor D, Saad Y. Modification and compensation strategies for threshold-based incomplete factorizations. *SIAM Journal on Scientific Computing* 2012; **34**(1):A48–A75, doi:10.1137/110834986. URL http://dx.doi.org/10.1137/110834986.
32. Saunders MA, Tomlin JA. Solving regularized linear programs using barrier methods and KKT systems. *Technical Report SOL-96-4*, SOL, Department of Operations Research, Stanford University 1996.
33. Sala M, Heroux M. Robust algebraic preconditioners with IFPACK 3.0. *Technical Report SAND-0662*, Sandia National Laboratories 2005.
34. Erlangga YA, Vuik C, Oosterlee CW. Comparison of multigrid and incomplete LU shifted-Laplace preconditioners for the inhomogeneous Helmholtz equation. *Applied Numerical Mathematics* 2006; **56**(5):648–666.
35. Osei-Kuffuor D, Saad Y. Preconditioning Helmholtz linear systems. *Applied Numerical Mathematics* 2010; **60**(4):420–431.
36. Bunch JR. Analysis of the diagonal pivoting method. *SIAM Journal on Numerical Analysis* 1971; **8**:656–680.
37. Druinsky A, Toledo S. The growth-factor bound for the Bunch-Kaufman factorization is tight. *SIAM Journal on Matrix Analysis and Applications* 2011; **32**(3):928–937.
38. Cormen TH, Leiserson CE, Rivest RL, Stein C. *Introduction to algorithms*. Third edn., MIT Press, Cambridge, MA, 2009.
39. Saad Y, Schulz MH. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing* 1986; **7**:856–869.
40. Scott JA, Tůma M. The importance of structure in incomplete factorization preconditioners. *BIT Numerical Mathematics* 2011; **51**:385–404.
41. Hogg JD, Scott JA. HSL_MA97: a bit-compatible multifrontal code for sparse symmetric systems. *Technical Report RAL-TR-2011-024*, Rutherford Appleton Laboratory 2011.
42. Davis TA, Hu Y. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software* 2011; **38**(1).

43. Duff IS, Koster J. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal on Matrix Analysis and Applications* 2001; **22**:973–996.
44. Duff IS, Pralet S. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM Journal on Matrix Analysis and Applications* 2005; **27**:313–340.
45. Hogg JD, Scott JA. The effects of scalings on the performance of a sparse symmetric indefinite solver. *Technical Report RAL-TR-2008-007*, Rutherford Appleton Laboratory 2008.

DOI: 10.1002/nla