SYM-ILDL: Incomplete LDL^{T} Factorization of Symmetric Indefinite and Skew-Symmetric Matrices

Chen Greif, Shiwen He, and Paul Liu, University of British Columbia

SYM-ILDL is a numerical software package that computes incomplete LDL^{T} (or 'ILDL') factorizations of symmetric indefinite and skew-symmetric matrices. The core of the algorithm is a Crout variant of incomplete LU (ILU), originally introduced and implemented for symmetric matrices by [Li and Saad, *Crout versions of ILU factorization with pivoting for sparse symmetric matrices*, Transactions on Numerical Analysis 20, pp. 75–85, 2005]. Our code is economical in terms of storage and it deals with skew-symmetric matrices as well, in addition to symmetric ones. The package is written in C++ and it is templated, open source, and includes a MATLAB™ interface. The code includes built-in RCM and AMD reordering, two equilibration strategies, threshold Bunch-Kaufman pivoting and rook pivoting, among other features. We also include an efficient MINRES implementation, applied with a specialized symmetric positive definite preconditioning technique based on the ILDL factorization.

1. INTRODUCTION

For the numerical solution of symmetric and skew-symmetric linear systems of the form

$$Ax = b,$$

stable (skew-)symmetry-preserving decompositions of A often have the form

$$PAP^T = LDL^T,$$

where L is a (possibly dense) lower triangular matrix and D is a block-diagonal matrix with 1-by-1 and 2-by-2 blocks [Bunch 1982; Bunch and Kaufman 1977]. The matrix P is a permutation matrix, satisfying $PP^T = I$, and the right-hand side vector b is permuted accordingly: in practice we solve $(PAP^T)(Px) = Pb$.

In the context of incomplete LDL^{T} (ILDL) decompositions of sparse and large matrices for preconditioned iterative solvers, various element-dropping strategies are commonly used to impose sparsity of the factor, L. Fill-reducing reordering strategies are also used to encourage the sparsity of L, and various scaling methods are applied to improve conditioning. For a symmetric linear system, several methods have been developed. Recently proposed approaches perturb or partition A so that incomplete Cholesky may be used [Lin and Moré 1999; Orban 2014; Scott and Tuma 2014a]. While [Lin and Moré 1999] is designed for positive definite matrices, the recent papers of Orban [2014] and Scott and Tuma [2014a] are applicable to a large set of 2×2 block structured indefinite systems.

We present SYM-ILDL — a software package based on a left-looking Crout version of LU, which is stabilized by pivoting strategies such as Bunch-Kaufman and rook pivoting. The algorithmic principles underlying our software are based on (and extend) an incomplete LDL^{T} factorization approach proposed by Li and Saad [2005], which itself extends work by [Li et al. 2003] and [Jones and Plassmann 1995]. We offer the following new contributions:

— A Crout-based incomplete LDL^{T} factorization for skew-symmetric matrices is introduced in this paper for the first time. It features a similar mechanism to the one for symmetric indefinite matrices, but there are notable differences. Most importantly, for skew-symmetric matrices the diagonal elements of D are zero and the pivots are always 2×2 blocks.

— We offer an integrated preconditioned MINRES solver, specialized to our ILDL code. The main challenge here is to design a positive definite preconditioner, even though ILDL produces an indefinite (or skew-symmetric) factorization. To that end, for the symmetric case we implement the technique presented in Gill et al. [1992]. For the skew-symmetric case we introduce a positive definite preconditioner based on exploiting the simple 2×2 structure of the pivots.

— The code is written in C++, and is templated and easily extensible. As such, it can be easily modified to work in other fields of numbers, such as $\mathbb{C}$. SYM-ILDL is self-contained and it includes implementations of reordering methods (AMD and RCM), equilibration methods (in the max-norm, 1-norm, and 2-norm), and pivoting methods (Bunch-Kaufman and rook pivoting). To facilitate ease of use, a MATLAB™ interface is provided.

Incomplete factorizations of symmetric indefinite matrices have received much attention recently and a few numerical packages have been developed in the past few years. Scott and Tuma [2014a] have developed a numerical software package based on signed incomplete Cholesky factorization preconditioners due to Lin and Moré [1999]. For saddle-point systems, Scott and Tuma [2014a] have extended their limited memory incomplete Cholesky algorithm [Scott and Tuma 2014b] to a signed incomplete Cholesky factorization. Their approach builds on the ideas of Tismenetsky [1991] and Kaporin [1998]. In the case of breakdown (a zero pivot), a global shift is applied (see also [Lin and Moré 1999]).

Scott and Tuma [2014a, Section 6.4] have made comparisons with our code, and have found that in general, the two codes are comparable in performance for several of the test problems, whereas for some of the problems each code outperforms the other. Given the comprehensive nature of the comparisons in Scott and Tuma [2014a], we do not provide further comparisons between the two packages in this paper.

Orban [2014] has developed `LLDL`, a generalization of the limited-memory Cholesky factorization of Lin and Moré [1999] to the symmetric indefinite case with special interest in symmetric quasi-definite matrices. The code generates a factorization of the form $\mathrm{LDL}^{\mathrm{T}}$ with D diagonal. We are currently engaged, jointly with Orban, in a comparison of our code to `LLDL`.

The remainder of this paper is structured as follows. In Section 2 we outline a Crout-based factorization for symmetric and skew-symmetric matrices, symmetry-preserving pivoting strategies, equilibration approaches and reordering strategies. In Section 3 we discuss how to modify the output of SYM-ILDL to produce a positive definite preconditioner for MINRES. In Section 4 we discuss the implementation of SYM-ILDL, and how the pivoting strategies of Section 2 may be efficiently implemented within SYM-ILDL's data structures. Finally, we show the performance of SYM-ILDL on some general (skew-)symmetric matrices and some saddle-point matrices in Section 5.

2. LDL AND ILDL FACTORIZATIONS

SYM-ILDL uses a Crout variant of LU factorization. To maintain stability, SYM-ILDL allows the user to choose one of two pivoting symmetry-preserving strategies: Bunch-Kaufman partial pivoting [Bunch and Kaufman 1977] (Bunch in the skew-symmetric case [Bunch 1982]) and rook pivoting. The details of the factorization and pivoting procedures, as well as simplifications for the skew-symmetric case, are provided in the following sections. See also [Duff 2009] for more details on the use of direct solvers for solving skew-symmetric matrices.

2.1. Crout-based factorizations

The Crout order is an attractive way for computing an ILDL factorization of symmetric or skew-symmetric matrices, because it naturally preserves structural symmetry, especially when dropping rules for the incomplete factorization are applied. As opposed to the IKJ-based approach [Li and Saad 2005], Crout relies on computing and applying dropping rules to a column of L and a row of U simultaneously. The Crout procedure for a symmetric matrix is outlined in Algorithm 2, using a delayed update procedure for the factors which is laid out in Algorithm 1. (As discussed in the sequel, the procedure in Algorithm 1 may be called multiple times when various pivoting procedures are employed.)

ALGORITHM 1: Factors update procedure

Input: A symmetric matrix A, partial factors L and D, matrix size n, current column index k
Output: Updated factors L and D

$L_{k:n,k} \leftarrow A_{k:n,k}$
$i \leftarrow 1$
while $i < k$ **do**
  $s_i \leftarrow$ size of the diagonal block with $D_{i,i}$ as its top left corner
  $L_{k:n,k} \leftarrow L_{k:n,k} - L_{k:n,i:i+s_i-1} D^{-1}_{i:i+s_i-1,i:i+s_i-1} L^T_{k,i:i+s_i-1}$
  $i \leftarrow i + s_i$
end

ALGORITHM 2: Crout factorization, LDL$^{\mathrm{T}}$C

Input: A symmetric matrix A
Output: Matrices P, L, and D, such that $PAP \approx LDL^T$

1 $k \leftarrow 1$
2 $L \leftarrow \mathbf{0}$
3 $D \leftarrow \mathbf{0}$
4 **while** $k < n$ **do**
5   Call Algorithm 1 to update L and D
6   Find a pivoting matrix in $A_{k:n,k:n}$ and permute A accordingly
7   $s \leftarrow$ size of the pivoting matrix
8   $D_{k:k+s-1,k:k+s-1} \leftarrow L_{k:k+s-1,k:k+s-1}$
9   $L_{k+s:n,k:k+s-1} \leftarrow L_{k+s:n,k:k+s-1} D^{-1}_{k:k+s-1,k:k+s-1}$
10   Apply dropping rules to $L_{k+s:n,k:k+s-1}$
11   $k \leftarrow k + s$
12 **end**

For computing the ILDL factorization, we apply dropping rules; see line 10 of Algorithm 2. These are the standard rules: we drop all entries below a pre-specified tolerance (referred to as `drop_tol` throughout the paper), multiplied by the norm of a column of L, keeping up to a pre-specified maximum number of the largest nonzero entries in every column. We use here the term `fill_factor` to signify the maximum allowed ratio between the number of nonzeros in any column of L and the average number of nonzeros per column of A.

In Algorithm 2, the $s \times s$ pivot is typically 1×1 or 2×2, as per the strategy devised by Bunch and Kaufman [1977], which we briefly describe next.

2.2. Symmetric partial pivoting

Pivoting in the symmetric or skew-symmetric setting is challenging, since we seek to preserve the (skew-)symmetry and it is not sufficient to use 1×1 pivots to maintain stability. Much work has been done in this front; see, for example, [Duff et al. 1989; Duff et al. 1991; Hogg and Scott 2014] and the references therein.

Bunch and Kaufman [1977] proposed a partial pivoting strategy for symmetric matrices, which relies on finding 1×1 and 2×2 pivots. The cost of finding a pivot is $\mathcal{O}(n)$, as it only involves searching up to two columns. We provide this procedure in Algorithm 3.

The constant $\alpha = (1 + \sqrt{17})/8$ in line 1 of the algorithm controls the growth factor, and a_{ij} is the ij-th entry of the matrix A after computing all the delayed updates in Algorithm 1 on column i. Although the partial pivoting strategy is backward stable [Higham 2002], the possibly large elements in the unit lower triangular matrix L may cause numerical difficulty. Rook pivoting provides an alternative that in practice proves to be more stable, at a modest additional cost. This procedure is presented in Algorithm 4. The algorithm searches the

ALGORITHM 3: Bunch-Kaufman LDL$^{\mathrm{T}}$ using partial pivoting strategy

$\alpha \leftarrow (1+\sqrt{17})/8 \ (\approx 0.64)$
$\omega_1 \leftarrow$ maximum magnitude of any subdiagonal entry in column 1
if $|a_{11}| \geq \alpha\omega_1$ **then**
    Use a_{11} as a 1×1 pivot ($s = 1$)
else
    $r \leftarrow$ row index of first (subdiagonal) entry of maximum magnitude in column 1
    $\omega_r \leftarrow$ maximum magnitude of any off-diagonal entry in column r
    if $|a_{11}|\omega_r \geq \alpha\omega_1^2$ **then**
        Use a_{11} as a 1×1 pivot ($s = 1$)
    else if $|a_{rr}| \geq \alpha\omega_r$ **then**
        Use a_{rr} as a 1×1 pivot ($s = 1$, swap rows and columns 1, r)
    else
        Use $\begin{pmatrix} a_{11} & a_{r1} \\ a_{r1} & a_{rr} \end{pmatrix}$ as a 2×2 pivot ($s = 2$, swap rows and columns 2, r)
    end
end

pivots of the matrix in spiral order until it finds an element that is largest in absolute value in both its row and its column, or terminates if it finds a relatively large diagonal element. Although theoretically rook pivoting could traverse many columns, we have found that it is fast in practice, and we use it as the default pivoting scheme of SYM-ILDL.

ALGORITHM 4: LDL$^{\mathrm{T}}$ using rook pivoting strategy

$\alpha \leftarrow (1+\sqrt{17})/8 \ (\approx 0.64)$
$\omega_1 \leftarrow$ maximum magnitude of any subdiagonal entry in column 1
if $|a_{11}| \geq \alpha\omega_1$ **then**
    Use a_{11} as a 1×1 pivot ($s = 1$)
else
    $i \leftarrow 1$
    while *a pivot is not yet chosen* **do**
        $r \leftarrow$ row index of first (subdiagonal) entry of maximum magnitude in column i
        $\omega_r \leftarrow$ maximum magnitude of any off-diagonal entry in column r
        if $|a_{rr}| \geq \alpha\omega_r$ **then**
            Use a_{rr} as a 1×1 pivot ($s = 1$, swap rows and columns 1 and r)
        else if $\omega_i = \omega_r$ **then**
            Use $\begin{pmatrix} a_{ii} & a_{ri} \\ a_{ri} & a_{rr} \end{pmatrix}$ as a 2×2 pivot ($s = 2$, swap rows and columns 1 and i, and 2 and r)
        else
            $i \leftarrow r$
            $\omega_i \leftarrow \omega_r$
        end
    end
end

2.3. Equilibration and reordering strategies

In many cases of practical interest, the input matrix is ill-conditioned. For these cases, equilibration schemes have been shown to be effective in lowering the condition number of the matrix. Symmetric equilibration schemes rescale entries of the matrix by computing a diagonal matrix D such that DAD has equal row norms and column norms.

SYM-ILDL offers two equilibration schemes: Bunch's equilibration in the max norm [Bunch 1971] and Ruiz's iterative equilibration in any L_p-norm [Ruiz 2001].

Bunch's equilibration allows the user to scale the max norm of every row and column to 1 before factorization. Let T be the lower triangular part of A in absolute value (diagonal included), that is, $T_{ij} = |A_{ij}|,\ 1 \le j \le i \le n$. Then Bunch's algorithm runs in $\mathcal{O}(\mathrm{nnz}(A))$ time, and is based on the following greedy procedure:
For $1 \le i \le n$, set

$$D_{ii} := \left(\max\left\{\sqrt{T_{ii}}, \max_{1\le j\le i-1} D_{jj}T_{ij}\right\}\right)^{-1}.$$

Ruiz's equilibration allows the user to scale every row and column of the matrix to 1 in any L_p norm, provided that $p \ge 1$ and the matrix has support [Ruiz 2001]. For the max norm, Ruiz's algorithm scales each column's norm to within ε of 1 in $\mathcal{O}(\mathrm{nnz}(A)\log\frac{1}{\varepsilon})$ time for any given tolerance ε.

Let $r(A,i)$ and $c(A,i)$ denote the i -th row and column of A respectively, and let $D(i,\alpha)$ to be the diagonal matrix with $D_{jj} = 1$ for all $j \ne i$ and $D_{ii} = \alpha$. Using this notation, our variant of Ruiz's algorithm is shown in Algorithm 5.

ALGORITHM 5: Equilibrating general matrices in the max-norm

Input: A general matrix A
Output: Diagonal matrices R and C such that RAC has max-norm 1 in every row and column

$R \leftarrow \mathbf{I}$
$C \leftarrow \mathbf{I}$
$\tilde{A} \leftarrow A$
while R **and** C *have not yet converged* **do**
    for i := *1 to* n **do**
        $\alpha_r \leftarrow \frac{1}{\sqrt{||r(\tilde{A},i)||_\infty}}$
        $\alpha_c \leftarrow \frac{1}{\sqrt{||c(\tilde{A},i)||_\infty}}$
        $R \leftarrow R \cdot D(i,\alpha_r)$
        $C \leftarrow C \cdot D(i,\alpha_c)$
        $\tilde{A} \leftarrow D(i,\alpha_r)\tilde{A}D(i,\alpha_c)$
    end
end

Our presentation differs from Ruiz's original algorithm in that it operates one row and column at a time as opposed to operating on the entire matrix in each iteration. We implemented the algorithm this way as it naturally adapts to our storage structures; our code is more easily amenable to single column operations rather than matrix-vector products. However, a proof of correctness similar to that of of Ruiz's algorithm applies, with the same guarantee for the running time.

Ruiz's strategy seems to perform well in terms of preserving diagonal dominance when no reordering strategy is used. In fact, we have observed that for certain skew-symmetric systems, Ruiz's equilibration leads to convergence of the iterative solver, while Bunch's approach does not. On the other hand, Bunch's equilibration strategy is faster, being a one-pass procedure. In our experiments we use Bunch as the default.

After equilibration, we carry out a reordering strategy. The user is given the option of choosing from Approximate Minimum Degree (AMD) [Amestoy et al. 1996] and Reverse Cuthill-McKee (RCM) [George and Liu 1981]. We have found AMD to be generally more effective for our test cases, and it is set as the default in the code.

2.4. LDL and ILDL factorizations for skew-symmetric matrices

The skew-symmetric case is different than the symmetric indefinite case in the sense that here, we must always use 2×2 pivots, because diagonal elements of skew-symmetric matrices are zero. This significantly simplifies the Bunch-Kaufman procedure: we have only one case rather than four. Algorithm 6 illustrates the simplification for rook pivoting. Furthermore, as opposed to a typical 2×2 symmetric matrix, which is defined by three parameters, the analogous skew-symmetric matrix is defined by one parameter only. As a result, at the kth step, the computation of the multiplier and the subsequent update of pair of columns associated with the pivoting operation can be expressed as follows:

$$
\begin{aligned}
A_{k+2:n,k:k+1}A^{-1}_{k:k+1,k:k+1} &= A_{k+2:n,k:k+1}\begin{pmatrix} 0 & -a_{k+1,k} \\ a_{k+1,k} & 0 \end{pmatrix}^{-1} \\
&= \frac{1}{a_{k+1,k}} A_{k+2:n,k:k+1}\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix},
\end{aligned}
$$

which can be trivially computed by swapping columns k and $k+1$ and scaling.

ALGORITHM 6: $\mathrm{LDL}^{\mathrm{T}}$ using rook pivoting strategy for skew-symmetric matrices

---

$\omega_1 \leftarrow$ maximum magnitude of any subdiagonal entry in column 1
$i \leftarrow 1$
while *a pivot is not yet chosen* **do**
    $r \leftarrow$ row index of first (subdiagonal) entry of maximum magnitude in column i
    $\omega_r \leftarrow$ maximum magnitude of any off-diagonal entry in column r
    if $\omega_i = \omega_r$ **then**
        U
    else
        s
    end
    e $\begin{pmatrix} 0 & -a_{ri} \\ a_{ri} & 0 \end{pmatrix}$ as a 2×2 pivot (swap rows and columns 1 and i, and 2 and r)
end
$i \leftarrow r$
$\omega_i \leftarrow \omega_r$

The ILDL factorization for skew-symmetric matrices can thus be carried out similarly to the manner in which it is developed for symmetric indefinite matrices, but the eventual algorithm gives rise to the above described simplifications. Skew-symmetric matrices are often ill-conditioned, and we have experimentally found that computing a numerical solution effectively for those systems is challenging. More details are provided in Section 5.

3. A SPECIALIZED PRECONDITIONER FOR MINRES

Our goal is to use a symmetric solver with our ILDL factorization, given that LDL^T is indefinite (for symmetric indefinite A) or skew-symmetric (for skew-symmetric A). The main difficulty lies in the fact that preconditioners for short recurrence solvers typically must be symmetric positive definite.

We describe below techniques for generating MINRES preconditioned iterations, using positive definite versions of the incomplete factorization. For the symmetric indefinite case, we apply the method presented in [Gill et al. 1992]. Given $M = LDL^T$, let us focus our attention on the various options for the blocks of D. Our ultimate goal is to modify D and L such that D is diagonal with only 1 or -1 as its diagonal entries. If a block of the matrix D

from the original LDL factorization was 2×2, then the corresponding modified (diagonal) block would become

$$\begin{pmatrix} \pm 1 & 0 \\ 0 & \mp 1 \end{pmatrix}.$$

For a diagonal entry of D that appears as a 1×1 block, say, $d_{i,i}$, we rescale the ith row of L: $L(i,:) \to L(i,:)\sqrt{|d_{i,i}|}$. We can then set the new value of $d_{i,i}$ as ± 1. In practice there is no need to perform a multiplication of a row of L by $\sqrt{|d_{i,i}|}$; instead, this scalar is stored separately and its multiplicative effect is computed as an $\mathcal{O}(1)$ operation for every matrix vector product.

Now, consider a 2×2 block of D, say D_j. For this case, we compute the eigendecomposition

$$D_j = Q_j \Lambda_j Q_j^T,$$

and similarly to the case of a 1×1 block, we *implicitly* rescale two rows of L by $Q_j\sqrt{|\Lambda_j|}$. This means that L is no longer triangular; it is in fact lower Hessenberg, since some values above the main diagonal may become nonzero. But the solve is just as straightforward, since the decomposition is explicitly given.

In the skew-symmetric case, we may use a specialized version of MINRES [Greif and Varah 2009]. We only have 2×2 blocks, and for those, we know that

$$\begin{pmatrix} 0 & a_{j,j} \\ -a_{j,j} & 0 \end{pmatrix} = \begin{pmatrix} \sqrt{|a_{j,j}|} & 0 \\ 0 & \sqrt{|a_{j,j}|} \end{pmatrix} \begin{pmatrix} 0 & \pm 1 \\ \mp 1 & 0 \end{pmatrix} \begin{pmatrix} \sqrt{|a_{j,j}|} & 0 \\ 0 & \sqrt{|a_{j,j}|} \end{pmatrix}.$$

Therefore, we do not need an eigendecomposition (as in the symmetric case), and instead we just scale the two affected rows of L by $\sqrt{|a_{j,j}|}I_2$.

Figure 1 shows the clustering effect that the proposed preconditioning approach has. We generate a symmetric random 300×300 matrix, say A, and compute the eigenvalues of $(LDL^T)^{-1}A$, where L and D are the matrices generated in the above described preconditioning procedure. Our fill factor is 2.0 and the drop tolerance was 10^{-4}. We note that the eigenvalue distribution in the figure is typical for other cases that were tested.

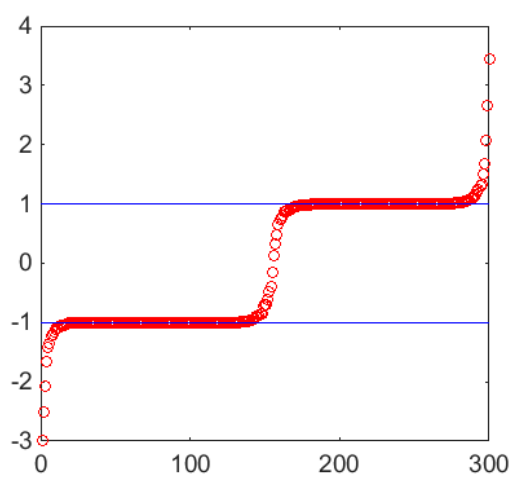

Fig. 1. Eigenvalues of a preconditioned symmetric random 300×300 matrix. The horizontal blue lines mark 1 and -1.

4. IMPLEMENTATION

4.1. Matrix storage in SYM-ILDL

Since we are dealing with symmetric or skew-symmetric matrices, one of our goals is to avoid duplicating data. At the same time, it is necessary for SYM-ILDL to have fast column access as well as fast row access. In terms of storage, we deal with these requirements by generating a format similar to standard compressed sparse column form, along with compressed sparse row form without the nonzero floating point matrix values. Matrices are stored in a *list-of-arrays* format. Each column is represented internally as two arrays, storing both its nonzero values `col_val` and row indices (`col_list`). One advantage of this format is that swapping columns and deallocating their memory is much easier. Additionally, a row-major data structure (`row_list`) is used to maintain fast access across the nonzeros of each row (see Figure 2). This is obtained by representing each row internally as a single array, storing the column indices of each row in an array (the nonzero values are already stored in the column-major representation).

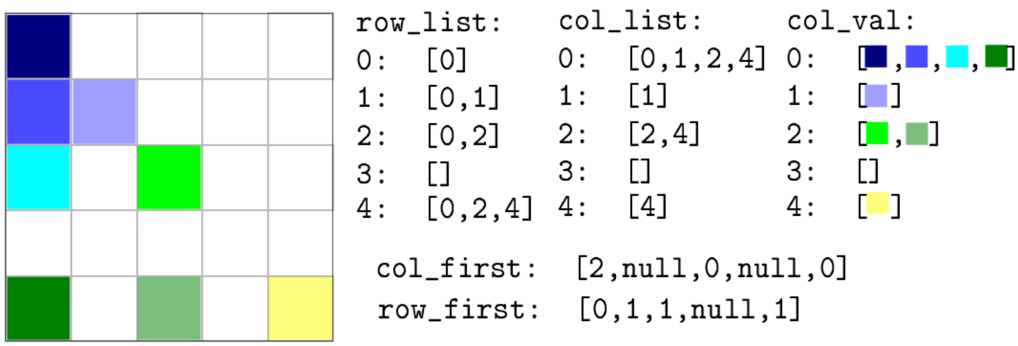

Fig. 2. Graphical representation of the data structures of SYM-ILDL. `col_first` and `row_first` are shown during the third iteration of the factorization. Hence `col_first` holds the values of indices in `col_list` for the first element under or on the third row of the matrix. Similarly, `row_first` holds the values of indices in `row_list` for the last element not exceeding the third column of the matrix.

Our format is a modest improvement over storing the full matrix in standard CSC, as used in [Li and Saad 2005]. Assuming that the row and column indices are stored in 32-bit integers and the nonzero values are stored in 64-bit doubles, this gives us an overall 33% saving in storage if we were to store the factorization in-place. This is an easy modification of Algorithm 3. In the default implementation, we find it more useful to store an equilibrated and permuted copy of the original matrix, so that we may use it for MINRES after the preconditioner is computed. An in-place version that returns only the preconditioner is included as part of our package.

4.2. Data structures for matrix access

In ILUC [Li and Saad 2005], a bi-index data structure was developed to address two implementation difficulties in sparse matrix operations, following earlier work by Eisenstat et al. [1981] in the context of the Yale Sparse Matrix Package (YSMP), and Jones and Plassmann [1995]. Our implementation uses a similar bi-index data structure, which we describe below.

Internally, the column and row indices in the matrix are stored in unsorted order. This avoids the cost of sorting whenever we need to pivot. Due to the unsortedness, accessing specific elements of the matrix is difficult and requires a slow linear search. Luckily, because Algorithm 3 accesses elements in a predictable fashion, we can speed up access to subcolumns required during the factorization to $\mathcal{O}(1)$ amortized time. The strategy we use to speed up matrix access is similar to that of [Jones and Plassmann 1995]. To ensure fast access to the submatrix $L_{k+1:n,1:k}$ and the row $L_{k,:}$ during factorization, we use one additional length n array: `col_first`. The i-th element of `col_first` array effectively holds a pointer to the start of the submatrix $L_{k+1:n,i}$ in `col_list` and speeds up Algorithm 1, allowing us access to the submatrix in $\mathcal{O}(1)$ time. To get fast access to the list of columns that contribute to the update of the $(k+1)$-st column, we use the row structure `row_list` discussed in section 4.1. Overall, this reduces the access time of the submatrix $L_{k+1:n,1:k}$ and row L_k down to a cost proportional to the number of nonzeros in these submatrices.

To ensure that `col_first(i)` points to the start of the subcolumn $L_{k+1:n,i}$ on step k, we advance the pointer for `col_first(i)` (if needed) at the end of processing the k-th column. Since the column indices in `col_list` are unsorted, this step requires a linear search to find the smallest element in `col_list`. Once this element is found, we swap it to the correct spot so that the column indices for $L_{k+1:n,i}$ are in a contiguous segment of memory.

Similarly, we will also need to access the subrows $A_{k,1:k}$ and $A_{r,1:k}$ during the pivoting stage (lines 11 to 15 in Algorithm 3 and Algorithm 4). This is sped up by an analogous `row_first(i)` structure that points to the end of the subrow $A_{i,1:k}$ ($A_{i,1:k}$ is the memory region that encompases everything from the start of memory for that row to `row_first(i)`). At the end of step k, we also advance the pointers for `row_first` if needed.

A summary of data structures can be found in Table I.

Table I. Variable names with data structure types

Variable name	**Data structure type**	**Purpose**
`col_first`	n length array	Speeds up access to $L_{k+1:n,i}$, i.e., `row_list`
`row_first`	n length array	Speeds up access to $A_{i,1:k}$, i.e., `col_list`
`row_list`	n linked lists (row-major)	Stores indices of A across the rows
`col_list`	n linked lists (col-major)	Stores indices of A across the columns
`col_val`	n linked lists (col-major)	Stores nonzero coefficients of A

5. NUMERICAL EXPERIMENTS

For testing our code, we use the University of Florida (UF) collection [Davis and Hu 2011], as well as our own matrices. The UF collection provides a variety of symmetric matrices, which we specify in Tables II and IV. We have used some of the same matrices that have been used in the papers [Li and Saad 2005; Li et al. 2003; Scott and Tuma 2014a].

We also test with simple discrete differential operators arising from two model problems. One is a model convection-diffusion equation, which is a discrete version of

$$-\Delta u + (\sigma, \tau, \mu)\nabla u = f, \tag{1}$$

with Dirichlet boundary conditions on the unit square, discretized using a uniform mesh of size h. We define the mesh Reynolds numbers $\beta = \sigma h/2, \gamma = \tau h/2, \delta = \mu h/2$. We use the skew-symmetric part of this matrix (that is, given A, form $\frac{A-A^T}{2}$) for our skew-symmetric experiments. Our second set are test matrices associated with the discrete Helmholtz equation,

$$-\Delta u - \alpha u = f, \tag{2}$$

subject to Dirichlet boundary conditions. Here we choose α so that a symmetric indefinite matrix is generated; see Table III.

All experiments were run on a single threaded, 2.8 GHZ AMD dual core machine, with 128 GB RAM. Timings are averaged across ten runs of each test case. In the experiments below, we follow the conventions of [Li and Saad 2005; Li et al. 2003] and define the *fill* of a factorization as $\mathrm{nnz}(L + D + L^T)/\mathrm{nnz}(A)$.

5.1. Results for symmetric matrices

In Table II we show the results of experiments with a set of matrices from [Davis and Hu 2011]. The matrix dimensions go up to approximately four million, with number of nonzeros going up to approximately 100 million. We show timings for constructing the ILDL factorization and an iterative solution, applying the preconditioned MINRES approach described in Section 3. We apply Bunch's equilibration and AMD reordering before generating the incomplete LDL factorization and running preconditioned MINRES. For the incomplete factorization, we apply rook pivoting. We observe a good convergence behavior for most of the matrices that were tested. We observe that for the matrices in this set, the computational time is $O(n \cdot \mathrm{nnz}(A))$, as expected.

Table II. Factorization timings and MINRES iterations for test matrices

matrix	n	nnz(A)	fill	time (s)	iterations
aug3dcqp	35543	128115	1.8	0.08	28
bloweya	30004	150009	1.1	0.05	18
bratu3d	27792	173796	3.7	0.35	80
tuma1	22967	87760	2.6	0.09	201
tuma2	12992	49365	2.6	0.05	149
boyd1	93279	1211231	0.9	0.09	3
brainpc2	27607	179395	1.1	0.48	48
mario001	38434	204912	3.4	0.44	146
qpband	20000	45000	1.1	0.01	3
G3-circuit	1585478	7660826	5.0	9.5	100
Hook-1498	59374451	64531701	3.9	240.1	242
StocF-1465	1465137	21005389	1.8	18.9	379
Geo_1438	60236322	8580313	5.0	428.6	26
Serena	64131971	48538952	4.8	386.3	22
nlpkkt80	28192672	14883536	6.2	2179	998
nlpkkt120	95117792	96845792	6.2	788.5	907

The experiments were run with `fill_factor` = 2.0 for the smaller matrices and `fill_factor` = 4.0 for matrices larger than one million in dimension. The tolerance was `drop_tol` = 10^{-4}, and we used rook pivoting to maintain stability. The iteration was terminated when the norm of the relative residual went below 10^{-6}.

In Table III we present results for the Helmholtz model problem. We compare SYM-ILDL to MATLAB's ILUTP. For ILUTP we used a drop tolerance of 10^{-3} in all test cases. For ILDL, the `fill_factor` was set to ∞ (since ILUTP does not limit its intermediate memory by a fill factor) while the `drop_tol` parameter was then chosen to get roughly the same fill as that of ILUTP. In the context of the ILUTP preconditioner, the *fill* is defined as $\mathrm{nnz}(L + U)/\mathrm{nnz}(A)$.

Since ILUTP produces an L and a U, GMRES was used as the iterative solver instead of MINRES for both ILDL and ILUTP. In the case of ILDL, LD and L^T were used as the two preconditioners. Note that during the computation of the preconditioner, the in-place version of ILDL uses only about 2/3 of the memory used by ILUTP. During the GMRES solve, the ILDL preconditioner only uses about 1/2 the memory used by ILUTP.

We observe that the performance of ILDL on the Helmholtz model problem is dependent on the value of α chosen, but that if ILDL is given the same memory resources as ILUTP, it outperforms it. For $\alpha = 0.3$, the ILDL approach leads to lower iteration counts even when approximately 1/2 of the memory is allocated (i.e., when the same fill is allowed), whereas for $\alpha = 0.7$, ILUTP outperforms ILDL when the fill is roughly the same. If we allow ILDL to have memory usage as large as ILUTP (i.e., up to twice the fill), we see that ILDL clearly has lower iteration counts for GMRES.

Table III. Comparison of MATLAB's ILUTP and SYM-ILDL for Helmholtz matrices

matrix	n	nnz(A)	ilu fill	ilu gmres iters	ildl fill	ildl gmres iters
$\alpha = 0.3$						
helmholtz80	6400	31680	8.0	14	7.9	12
helmholtz120	14400	71520	10.8	17	10.9	6
helmholtz160	25600	127360	13.3	20	12.1	10
helmholtz200	40000	199200	16.6	39	13.4	15
$\alpha = 0.7$						
helmholtz80	6400	31680	9.8	11	9.9	28
helmholtz120	14400	71520	13.4	10	13.3	144
helmholtz160	25600	127360	19.9	18	19.4	84
helmholtz200	40000	199200	22.8	28	25.4	144
$\alpha = 0.7$, Equal memory for ILDL and ILUTP						
helmholtz80	6400	31680	9.8	11	13.6	6
helmholtz120	14400	71520	13.4	10	22.8	6
helmholtz160	25600	127360	19.9	18	29.9	9
helmholtz200	40000	199200	22.8	28	35.5	16

The parameter α in Equation 2 is indicated above. GMRES was terminated when the relative residual decreased below 10^{-6}.

In Table IV we compare our code to the code of Li et al. [2003], which was provided to us by Saad. We will refer to this code as LSC-ILDL. We use the same basic algorithms, but our approach saves on memory thanks to exploiting symmetry. One bottleneck for the code of Li et al. [2003] is input reading. On the other hand, that code is faster than our code for the factorization, as the entire input matrix is stored. Altogether, we observe SYM-ILDL has a slight edge in terms of overall computational time.

Table IV. Time comparisons between the code from Li et al. [2003] and ours

Matrix name	n	nnz(A)	fill	LSC-ILDL	SYM-ILDL
tuma1	22967	87760	1.81	0.06	0.04
bratu3d	27792	173796	2.07	0.10	0.10
stokes128	49666	558594	1.50	0.36	0.18
d_pretok	182730	1641672	1.73	1.20	1.07
turon_m	189924	1690876	1.76	1.25	1.17
darcy003	389874	2101242	2.04	1.79	1.51
c-big	345241	2341011	1.25	1.63	1.45
maxwell7	523265	5062950	1.26	4.39	3.72

All matrices were run with `fill_factor` $= 1.0$, `drop_tol` $= 10^{-4}$, and used Bunch-Kaufman partial pivoting to maintain stability. The two right-most columns report computational times in seconds.

5.2. Results for skew-symmetric matrices

We now report on the performance of SYM-ILDL for skew-symmetric matrices. We have tested on the skew-symmetric part of the model convection-diffusion equation (1). We have found that for the matrices we have tested, equilibration has not been particularly effective. We speculate that this might have to do with a property related to block diagonal dominance that these matrices have for certain values of the convective coefficients. Specifically, the norm of the tridiagonal part of the matrix is significantly larger than the norm of the remaining part. Equilibration tends to adversely affect this property by scaling down entries near the diagonal, and as a result the performance of an iterative solver often degrades. We thus do not apply equilibration in our skew-symmetric solver.

In Table V we manipulate the drop tolerance for ILDL, to obtain a fill nearly equal to that of ILUTP. For the latter we fix the drop tolerance at 0.001. This is done for the purpose of comparing the performance of the iterative solvers, when the memory requirements of ILUTP and ILDL are similar. Note, though, that our ILDL still consumes only about 2/3 of the memory of ILUTP, due to the fact that the floating point entries of only half of the matrix are stored. We see that the iteration counts are significantly better for ILDL, especially when rook pivoting is used.

Table V. Comparison of MATLAB's ILUTP and SYM-ILDL for a skew-symmetric matrix arising from a model convection-diffusion equation

n	nnz(A)	method	drop tol	fill	GMRES(100)
		ILDL-rook	4e−4	7.008	6
$20^3 = 8000$	45600	ILDL-partial	5e−4	6.861	6
		ILUTP	1e−3	7.758	8
		ILDL-rook	2e−4	10.973	8
$30^3 = 27000$	156600	ILDL-partial	3e−4	11.235	10
		ILUTP	1e−3	11.758	13
		ILDL-rook	9e−5	15.205	9
$40^3 = 64000$	374400	ILDL-partial	3e−4	15.686	18
		ILUTP	1e−3	15.654	19
		ILDL-rook	2e−5	21.560	6
$50^3 = 125000$	735000	ILDL-partial	2e−4	22.028	36
		ILUTP	1e−3	22.691	31
		ILDL-rook	2e−5	22.595	9
$60^3 = 216000$	1274400	ILDL-partial	4e−4	22.899	NC
		ILUTP	1e−3	23.483	70
		ILDL-rook	5e−6	32.963	5
$70^3 = 343000$	2028600	ILDL-partial	–	–	–
		ILUTP	1e−3	33.861	61

The parameter used were $\beta = 20, \gamma = 2, \delta = 1$. The MATLAB ILUTP used a drop tolerance of 0.001. 'NC' stands for 'no convergence'.

In Figure 3 we show the (complex) eigenvalues of the preconditioned matrix $(LDL^T)^{-1}A$, where A is the skew-symmetric part of 1 with convective coefficients $(\beta, \gamma, \delta) = (0.4, 0.5, 0.6)$, and LDL^T is the preconditioner generated by running SYM-ILDL with a drop tolerance of 10^{-3} and a fill-in parameter of 20. As seen in the figure, most of the eigenvalues are very strongly clustered around 1, which indicates that a preconditioned iterative solver is expected to rapidly converge.

6. CONCLUSION

We have presented SYM-ILDL, a C++ software package for solving and preconditioning symmetric or skew-symmetric matrices. Our algorithmic approach is based on that of Li and Saad [2005]. Our code extends the functionality of the code of Li and Saad [2005] by adding multiple pivoting, reordering, and equilibration schemes. For ease of use, the code is templated and offers a MATLAB™ interface. SYM-ILDL is open source, and can be found at `http://www.cs.ubc.ca/~greif/code/sym-ildl.html`. To facilitate the use of the factorization as a preconditioner for a symmetric solver, we apply a symmetric positive definite variant of the output factors of SYM-ILDL, which is similar to Gill et al. [1992] in the symmetric case. For the skew-symmetric case we derive a positive definite preconditioner which takes advantage of the simple nonzero structure of the 2×2 pivots.

Numerical results for SYM-ILDL and comparisons with the code of Li et al. [2003] and the widely used ILUTP preconditioner (as implemented by MATLAB™) indicate that for symmetric matrices, SYM-ILDL with rook pivoting is faster.

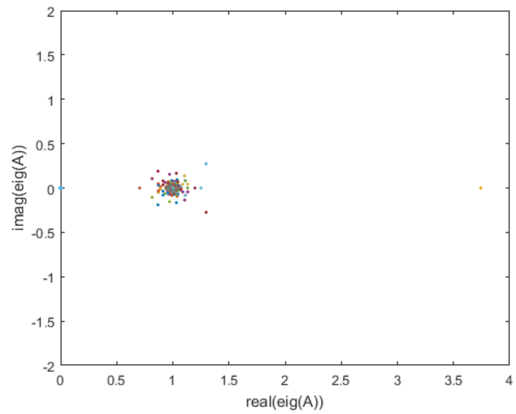

Fig. 3. Eigenvalues of a preconditioned skew-symmetric 1000×1000 matrix A arising from a convection-diffusion model problem.

More code optimization is possible, such as parallelization; such tasks remain as items for future work.

Acknowledgments

We would like to thank Dominique Orban and Jennifer Scott for their careful reading of an earlier version of this manuscript and for many helpful suggestions. We would also like to thank Yousef Saad for providing us with the code from [Li et al. 2003].

References

Patrick R. Amestoy, Timothy A. Davis, and Iain S. Duff. 1996. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.* 17, 4 (1996), 886–905. DOI:http://dx.doi.org/10.1137/S0895479894278952

James R. Bunch. 1971. Equilibration of Symmetric Matrices in the Max-Norm. *J. ACM* 18, 4 (Oct. 1971), 566–572. DOI:http://dx.doi.org/10.1145/321662.321670

James R Bunch. 1982. A note on the stable decompostion of skew-symmetric matrices. *Math. Comp.* 38, 158 (1982), 475–479.

James R. Bunch and Linda Kaufman. 1977. Some stable methods for calculating inertia and solving symmetric linear systems. *Math. Comp.* 31, 137 (1977), 163–179.

Timothy A. Davis and Yifan Hu. 2011. The University of Florida sparse matrix collection. *ACM Trans. Math. Software* 38, 1 (2011), Art. 1, 25. DOI:http://dx.doi.org/10.1145/2049662.2049663

Iain S. Duff. 2009. The design and use of a sparse direct solver for skew symmetric matrices. *J. Comput. Appl. Math.* 226, 1 (2009), 50–54. DOI:http://dx.doi.org/10.1016/j.cam.2008.05.016

I. S. Duff, A. M. Erisman, and J. K. Reid. 1989. *Direct methods for sparse matrices* (second ed.). The Clarendon Press, Oxford University Press, New York. xiv+341 pages. Oxford Science Publications.

I. S. Duff, N. I. M. Gould, J. K. Reid, J. A. Scott, and K. Turner. 1991. The factorization of sparse symmetric indefinite matrices. *IMA J. Numer. Anal.* 11, 2 (1991), 181–204. DOI:http://dx.doi.org/10.1093/imanum/11.2.181

Stanley C. Eisenstat, Martin H. Schultz, and Andrew H. Sherman. 1981. Algorithms and data structures for sparse symmetric Gaussian elimination. *SIAM J. Sci. Statist. Comput.* 2, 2 (1981), 225–237. DOI:http://dx.doi.org/10.1137/0902019

Alan George and Joseph W. H. Liu. 1981. *Computer solution of large sparse positive definite systems.* Prentice-Hall Inc., Englewood Cliffs, N.J. xii+324 pages. Prentice-Hall Series in Computational Mathematics.

Philip E. Gill, Walter Murray, Dulce B. Ponceleón, and Michael A. Saunders. 1992. Preconditioners for Indefinite Systems Arising in Optimization. *SIAM J. Matrix Anal. Appl.* 13, 1 (1992), 292–311. DOI:http://dx.doi.org/10.1137/0613022

Chen Greif and James M. Varah. 2009. Iterative solution of skew-symmetric linear systems. *SIAM J. Matrix Anal. Appl.* 31, 2 (2009), 584–601.

Nicholas J. Higham. 2002. *Accuracy and Stability of Numerical Algorithms* (2nd ed.). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

J. D. Hogg and J. A. Scott. 2014. Compressed threshold pivoting for sparse symmetric indefinite systems. *SIAM J. Matrix Anal. Appl.* 35, 2 (2014), 783–817. DOI:http://dx.doi.org/10.1137/130920629

Mark T. Jones and Paul E. Plassmann. 1995. An improved incomplete Cholesky factorization. *ACM Trans. Math. Software* 21, 1 (1995), 5–17. DOI:http://dx.doi.org/10.1145/200979.200981

Igor E. Kaporin. 1998. High quality preconditioning of a general symmetric positive definite matrix based on its $U^{\mathsf{T}}U + U^{\mathsf{T}}R + R^{\mathsf{T}}U$-decomposition. *Numer. Linear Algebra Appl.* 5, 6 (1998), 483–509 (1999). DOI:http://dx.doi.org/10.1002/(SICI)1099-1506(199811/12)5:6<483::AID-NLA156>3.3.CO;2-Z

Na Li and Yousef Saad. 2005. Crout versions of ILU factorization with pivoting for sparse symmetric matrices. *Electron. Trans. Numer. Anal.* 20 (2005), 75–85.

Na Li, Yousef Saad, and Edmond Chow. 2003. Crout versions of ILU for general sparse matrices. *SIAM J. Sci. Comput.* 25, 2 (2003), 716–728 (electronic). DOI:http://dx.doi.org/10.1137/S1064827502405094

Chih-Jen Lin and Jorge J. Moré. 1999. Incomplete Cholesky factorizations with limited memory. *SIAM J. Sci. Comput.* 21, 1 (1999), 24–45 (electronic). DOI:http://dx.doi.org/10.1137/S1064827597327334

Dominique Orban. 2014. Limited-memory $\mathrm{LDL}^{\mathrm{T}}$ factorization of symmetric quasi-definite matrices with application to constrained optimization. *Numerical Algorithms* (2014), 1–33. DOI:http://dx.doi.org/10.1007/s11075-014-9933-x

Daniel Ruiz. 2001. *A scaling algorithm to equilibrate both rows and columns norms in matrices*. Technical Report RAL-TR-2001-034. ENSEEIHT.

J. A. Scott and M. Tuma. 2014a. On Signed Incomplete Cholesky Factorization Preconditioners for Saddle-Point Systems. *SIAM J. Sci. Comput.* 36, 6 (2014a), A2984–A3010. DOI:http://dx.doi.org/10.1137/140956671

J. A. Scott and M. Tuma. 2014b. On positive semidefinite modification schemes for incomplete Cholesky factorization. *SIAM Journal on Scientific Computing* 36, 2 (2014b), A609–A633.

M. Tismenetsky. 1991. A new preconditioning technique for solving large sparse linear systems. *Linear Algebra Appl.* 154/156 (1991), 331–353. DOI:http://dx.doi.org/10.1016/0024-3795(91)90383-8