Modifying a Sparse Cholesky Factorization*

Timothy A. Davis†
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 32611
and
William W. Hager‡
Department of Mathematics
University of Florida, Gainesville, FL 32611

April 28, 1997

Technical Report TR-97-003, Department of Computer and Information Science and Engineering, University of Florida.

Abstract

Given a sparse symmetric positive definite matrix $\mathbf{A}\mathbf{A}^\mathsf{T}$ and an associated sparse Cholesky factorization $\mathbf{L}\mathbf{L}^\mathsf{T}$, we develop sparse techniques for obtaining the new factorization associated with either adding a column to $\mathbf{A}$ or deleting a column from $\mathbf{A}$. Our techniques are based on an analysis and manipulation of the underlying graph structure and on ideas of Gill, Golub, Murray, and Saunders for modifying a dense Cholesky factorization. Our algorithm involves a new sparse matrix concept, the multiplicity of an entry in $\mathbf{L}$. The multiplicity is essentially a measure of the number of times an entry is modified during symbolic factorization. We show that our methods extend to the general case where an arbitrary sparse symmetric positive definite matrix is modified. Our methods are optimal in the sense that they take time proportional to the number of nonzero entries in $\mathbf{L}$ that change.

*This work was supported by National Science Foundation grants DMS-9404431 and DMS-9504974.

†davis@cise.ufl.edu, http://www.cise.ufl.edu/~davis
‡hager@math.ufl.edu, http://www.math.ufl.edu/~hager

Key words. numerical linear algebra, direct methods, Cholesky factorization, sparse matrices, mathematical software, matrix updates.

AMS(MOS) subject classifications 65F05, 65F50, 65-04.

1 Introduction

This paper presents a method for updating and downdating the sparse Cholesky factorization $\mathbf{LL}^\mathsf{T}$ of the matrix $\mathbf{AA}^\mathsf{T}$, where $\mathbf{A}$ is m by n. More precisely, we evaluate the Cholesky factorization of $\mathbf{AA}^\mathsf{T} + \sigma\mathbf{ww}^\mathsf{T}$ where either σ is $+1$ (corresponding to an update) and $\mathbf{w}$ is arbitrary, or σ is -1 (corresponding to a downdate) and $\mathbf{w}$ is a column of $\mathbf{A}$. Both $\mathbf{AA}^\mathsf{T}$ and $\mathbf{AA}^\mathsf{T} + \sigma\mathbf{ww}^\mathsf{T}$ must be symmetric and positive definite. The techniques we develop for the matrix $\mathbf{AA}^\mathsf{T}$ can be extended to determine the effects on the Cholesky factors of a general symmetric positive definite matrix $\mathbf{M}$ of any symmetric change of the form $\mathbf{M} + \sigma\mathbf{ww}^\mathsf{T}$ that preserves positive definiteness. Our methods are optimal in the sense that they take time proportional to the number of nonzero entries in $\mathbf{L}$ that change.

There are many applications of the techniques presented in this paper. In the Linear Program Dual Active Set Algorithm (LP DASA) [19], the $\mathbf{A}$ matrix corresponds to the basic variables in the current basis of the linear program, and in successive iterations, we bring variables in and out of the basis, leading to changes of the form $\mathbf{AA}^\mathsf{T} + \sigma\mathbf{ww}^\mathsf{T}$. Other application areas where the techniques developed in this paper are applicable include least-squares problems in statistics, the analysis of electrical circuits, structural mechanics, sensitivity analysis in linear programming, boundary condition changes in partial differential equations, domain decomposition methods, and boundary element methods. For a discussion of these application areas and others, see [18].

Section 2 introduces our notation. For an introduction to sparse matrix techniques, see [4, 8]. In §3 we discuss the structure of the nonzero elements in the Cholesky factorization of $\mathbf{AA}^\mathsf{T}$ and in §4, we discuss the structure associated with the Cholesky factors of $\mathbf{AA}^\mathsf{T} + \sigma\mathbf{ww}^\mathsf{T}$. The symbolic update and downdate methods provide the framework for our sparse version of Method C5 of Gill, Golub, Murray, and Saunders [15] for modifying a dense Cholesky factorization. We discuss our sparse algorithm in §5. Section 6 presents the general algorithm for modifying the sparse Cholesky factorization for *any* sparse symmetric positive definite matrix. Implementation details for the modification algorithm associated with $\mathbf{AA}^\mathsf{T} + \sigma\mathbf{ww}^\mathsf{T}$ are summarized in §7, while the results of a numerical experiment with a large optimization problem from Netlib [3] are presented in §8. Section 9 concludes with a discussion of future work.

2 Notation

Throughout the paper, matrices are capital bold letters like $\mathbf{A}$ or $\mathbf{L}$, while vectors are lower case bold letters like $\mathbf{x}$ or $\mathbf{v}$. Sets and multisets are in calligraphic style like $\mathcal{A}$, $\mathcal{L}$, or $\mathcal{P}$. Scalars are either lower case Greek letters, or italic style like σ, k, or m.

Given the location of the nonzero elements of $\mathbf{A}\mathbf{A}^\mathsf{T}$, we can perform a *symbolic factorization* (this terminology is introduced by George and Liu in [8]) of the matrix to predict the location of the nonzero elements of the Cholesky factor $\mathbf{L}$. In actuality, some of these predicted nonzeros may be zero due to numerical cancellation during the factorization process. The statement "$l_{ij} \neq 0$" will mean that l_{ij} is *symbolically* nonzero. The diagonal of $\mathbf{L}$ is always nonzero since the matrices that we factor are positive definite (see [24, p. 253]). The nonzero pattern of column j of $\mathbf{L}$ is denoted $\mathcal{L}_j$:

$$\mathcal{L}_j = \{i \,:\, l_{ij} \neq 0\},$$

while $\mathcal{L}$ denotes the collection of patterns:

$$\mathcal{L} = \{\mathcal{L}_1, \mathcal{L}_2, \cdots, \mathcal{L}_m\}.$$

Similarly, $\mathcal{A}_j$ denotes the nonzero pattern of column j of $\mathbf{A}$,

$$\mathcal{A}_j = \{i \,:\, a_{ij} \neq 0\},$$

while $\mathcal{A}$ is the collection of patterns:

$$\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_n\}.$$

The *elimination tree* can be defined in terms of a *parent map* π (see [20]), which gives the row index associated with a given node j of the first nonzero element in column j of $\mathbf{L}$ beneath the diagonal element:

$$\pi(j) = \min\, \mathcal{L}_j \setminus \{j\},$$

where "min $\mathcal{X}$" denotes the smallest element of $\mathcal{X}$:

$$\min\, \mathcal{X} = \min_{i \in \mathcal{X}}\, i.$$

Our convention is that the min of the empty set is zero. Note that $j < \pi(j)$ except in the case where the diagonal element in column j is the only nonzero element. The inverse π^{-1} of the parent map is the *children multifunction*. That is, the children of node k is the set defined by

$$\pi^{-1}(k) = \{j : \pi(j) = k\}.$$

The *ancestors* of a node j, denoted $\mathcal{P}(j)$, is the set of successive parents:

$$\mathcal{P}(j) = \{j,\ \pi(j),\ \pi(\pi(j)),\ \cdots\} = \{\pi^0(j),\ \pi^1(j),\ \pi^2(j),\ \cdots\}.$$

Here the powers of a map are defined in the usual way: π^0 is the identity while π^i for $i > 0$ is the i-fold composition of π with itself. The sequence of nodes j, $\pi(j)$, $\pi(\pi(j))$, $\cdots$, forming $\mathcal{P}(k)$, is called the *path* from j to the associated tree *root*. The collection of paths leading to a root form an *elimination tree*, and the set of all trees is the *elimination forest*. Typically, there is a single tree whose root is m, however, if column j of $\mathbf{A}\mathbf{A}^\mathsf{T}$ has only one nonzero element, the diagonal element, then j will be the root of a separate tree.

The number of elements (or size) of a set $\mathcal{X}$ is denoted $|\mathcal{X}|$, while $|\mathcal{A}|$ or $|\mathcal{L}|$ denote the sum of the sizes of the sets they contain. Define the directed graph $G(\mathbf{M})$ of an m by m matrix $\mathbf{M}$ with nonzero pattern $\mathcal{M}$ as the vertex and edge sets: $G(\mathbf{M}) = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \{1, 2, \cdots, m\}$ is the vertex set and $\mathcal{E} = \{(i, j) \mid i \in \mathcal{M}_j\}$ is the directed edge set.

3 Symbolic factorization

Any approach for generating the pattern set $\mathcal{L}$ is called *symbolic factorization* [5, 6, 7, 8, 23]. The symbolic factorization of a matrix of the form $\mathbf{A}\mathbf{A}^\mathsf{T}$ is given in Algorithm 1 (see [9, 20]).

Algorithm 1 (Symbolic factorization of $\mathbf{A}\mathbf{A}^\mathsf{T}$)

$\pi(j) = 0$ for each j
for $j = 1$ **to** m **do**

$$\mathcal{L}_j = \{j\} \cup \left(\bigcup_{c \in \pi^{-1}(j)} \mathcal{L}_c \setminus \{c\} \right) \cup \left(\bigcup_{\min\ \mathcal{A}_k = j} \mathcal{A}_k \right)$$

$$\pi(j) = \min\ \mathcal{L}_j \setminus \{j\}$$

end for

Algorithm 1 basically says that the pattern of column j of $\mathcal{L}$ can be expressed as the union of the patterns of each column of $\mathbf{L}$ whose parent is j and the patterns of the columns of $\mathbf{A}$ whose first nonzero element is j. The elimination tree, connecting each child to its parent, is easily formed during the symbolic factorization. Algorithm 1 can be done in $O(|\mathcal{L}| + |\mathcal{A}|)$ time[1] [9, 20].

[1] Asymptotic complexity notation is defined in [2]. We write $f(n) = O(g(n))$ if there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n > n_0$.

Observe that the pattern of the parent of node j contains all entries in the pattern of column j except j itself [22]. That is,

$$\mathcal{L}_j \setminus \{j\} = \mathcal{L}_j \cap \{i : \pi(j) \leq i\} \subseteq \mathcal{L}_{\pi(j)}.$$

Proceeding by induction, if k is an ancestor of j, then

$$\{i : i \in \mathcal{L}_j, k \leq i\} \subseteq \mathcal{L}_k. \tag{1}$$

This leads to the following relation between $\mathcal{L}_j$ and the path $\mathcal{P}(j)$. The first part of this proposition, and its proof, is given in [22]. Our proof differs slightly from the one in [22]. We include it here since the same proof technique is exploited later.

Proposition 3.1 *For each j, we have $\mathcal{L}_j \subseteq \mathcal{P}(j)$; furthermore, for each k and $j \in \mathcal{P}(k)$, $\mathcal{L}_j \subseteq \mathcal{P}(k)$.*

Proof. Obviously, $j \in \mathcal{P}(j)$. Let i be any given element of $\mathcal{L}_j$ with $i \neq j$. Since $j < i$, we see that the following relation holds for $l = 0$:

$$\pi^0(j) < \pi^1(j) < \cdots < \pi^l(j) < i. \tag{2}$$

Now suppose that (2) holds for some integer $l \geq 0$, and let k denote $\pi^l(j)$. By (1) and the fact that $k \leq i$, we have $i \in \mathcal{L}_k$, which implies that

$$i \geq \pi(k) = \pi(\pi^l(j)) = \pi^{l+1}(j).$$

Hence, either $i = \pi^{l+1}(j)$ or (2) holds with l replaced by $l + 1$. Since (2) is violated for l sufficiently large, we conclude that there exists an l for which $i = \pi^{l+1}(j)$. Consequently, $i \in \mathcal{P}(j)$. Since each element of $\mathcal{L}_j$ is contained in $\mathcal{P}(j)$, we have $\mathcal{L}_j \subseteq \mathcal{P}(j)$. If $j \in \mathcal{P}(k)$ for some k, then j is an ancestor of k, and $\mathcal{P}(j) \subseteq \mathcal{P}(k)$. Since we have already shown that $\mathcal{L}_j \subseteq \mathcal{P}(j)$, the proof is complete. □

As we will see, the symbolic factorization of $\mathbf{A}\mathbf{A}^\mathsf{T} + \mathbf{w}\mathbf{w}^\mathsf{T}$ can be obtained by updating the symbolic factorization of $\mathbf{A}\mathbf{A}^\mathsf{T}$ using an algorithm that has the same structure as that of Algorithm 1. The new pattern $\overline{\mathcal{L}}_j$ is equal to the old pattern $\mathcal{L}_j$ union entries that arise from new children and from the pattern of the new column $\mathbf{w}$. (We put a bar over a matrix or a set or a multiset to denote its value after the update or downdate is complete.)

Downdating is not as easy. Once a set union has been computed, it cannot be undone without knowledge of how entries entered the set. We can keep track of this information by storing the elements of $\mathcal{L}$ as multisets rather than as sets. The multiset associated with column j has the form

$$\mathcal{L}_j^\sharp = \{(i, m(i, j)) : i \in \mathcal{L}_j\},$$

where the multiplicity $m(i,j)$ is the number of children of j that contain row index i in their pattern plus the number of columns of $\mathcal{A}$ whose smallest entry is j and that contain row index i. Equivalently,

$$m(i,j) = |\{k \in \pi^{-1}(j) : i \in \mathcal{L}_k\}| + |\{k : \min\ \mathcal{A}_k = j \text{ and } i \in \mathcal{A}_k\}|.$$

With this definition, we can undo a set union by subtracting multiplicities.

We now define some operations involving multisets. First, if $\mathcal{X}^\sharp$ is a multiset consisting of pairs $(i, m(i))$ where $m(i)$ is the multiplicity associated with i, then $\mathcal{X}$ is the set obtained by removing the multiplicities. In other words, the multiset $\mathcal{X}^\sharp$ and the associated base set $\mathcal{X}$ satisfy the relation:

$$\mathcal{X}^\sharp = \{(i, m(i)) : i \in \mathcal{X}\}.$$

We define the addition of a multiset $\mathcal{X}^\sharp$ and a set $\mathcal{Y}$ in the following way:

$$\mathcal{X}^\sharp + \mathcal{Y} = \{(i, m'(i)) \ : \ i \in \mathcal{X} \text{ or } i \in \mathcal{Y}\},$$

where

$$m'(i) = \begin{cases} 1 & \text{if } i \notin \mathcal{X} \text{ and } i \in \mathcal{Y}, \\ m(i) & \text{if } i \in \mathcal{X} \text{ and } i \notin \mathcal{Y}, \\ m(i)+1 & \text{if } i \in \mathcal{X} \text{ and } i \in \mathcal{Y}. \end{cases}$$

Similarly, the subtraction of a set $\mathcal{Y}$ from a multiset $\mathcal{X}^\sharp$ is defined by

$$\mathcal{X}^\sharp - \mathcal{Y} = \{(i, m'(i)) \ : \ i \in \mathcal{X} \text{ and } m'(i) > 0\},$$

where

$$m'(i) = \begin{cases} m(i) & \text{if } i \notin \mathcal{Y}, \\ m(i)-1 & \text{if } i \in \mathcal{Y}. \end{cases}$$

The multiset subtraction of $\mathcal{Y}$ from $\mathcal{X}^\sharp$ undoes a prior addition. That is, for any multiset $\mathcal{X}^\sharp$ and any set $\mathcal{Y}$, we have

$$((\mathcal{X}^\sharp + \mathcal{Y}) - \mathcal{Y}) = \mathcal{X}^\sharp.$$

In contrast $((\mathcal{X} \cup \mathcal{Y}) \setminus \mathcal{Y})$ is equal to $\mathcal{X}$ if and only if $\mathcal{X}$ and $\mathcal{Y}$ are disjoint sets.

Algorithm 2 below performs a symbolic factorization of $\mathbf{A}\mathbf{A}^\mathsf{T}$, with each set union operation replaced by a multiset addition. This algorithm is identical to Algorithm 1 except for the bookkeeping associated with multiplicities.

Algorithm 2 (Symbolic factorization of $\mathbf{A}\mathbf{A}^\mathsf{T}$, using multisets)

$\pi(j) = 0$ for each j

for $j = 1$ **to** m **do**

$\mathcal{L}_j^\sharp = \{(j,1)\}$
for each $c \in \pi^{-1}(j)$ **do**
$\mathcal{L}_j^\sharp = \mathcal{L}_j^\sharp + (\mathcal{L}_c \setminus \{c\})$
end for
for each k where min $\mathcal{A}_k = j$ **do**
$\mathcal{L}_j^\sharp = \mathcal{L}_j^\sharp + \mathcal{A}_k$
end for
$\pi(j) = \min\ \mathcal{L}_j \setminus \{j\}$
end for

We conclude this section with a result concerning the relation between the patterns of $\mathbf{A}\mathbf{A}^\mathsf{T}$ and the patterns of $\mathbf{A}\mathbf{A}^\mathsf{T} + \mathbf{w}\mathbf{w}^\mathsf{T}$.

Proposition 3.2 *Let $\mathcal{C}$ and $\mathcal{D}$ be the patterns associated with the symmetric positive definite matrices $\mathbf{C}$ and $\mathbf{D}$ respectively. Neglecting numerical cancellation, $\mathcal{C}_j \subseteq \mathcal{D}_j$ for each j implies that $(\mathcal{L}_C)_j \subseteq (\mathcal{L}_D)_j$ for each j, where $\mathcal{L}_C$ and $\mathcal{L}_D$ are the patterns associated with the Cholesky factors of $\mathbf{C}$ and $\mathbf{D}$ respectively.*

Proof. In [8, 21] it is shown that an edge (i,j) is contained in the graph of the Cholesky factor of a symmetric positive definite matrix $\mathbf{C}$ if and only if there is a path from i to j in the graph of $\mathbf{C}$ with each intermediate vertex of the path between 1 and min $\{i,j\}$. If $\mathcal{C}_j \subseteq \mathcal{D}_j$ for each j, then the paths associated with the graph of $\mathbf{C}$ are a subset of the paths associated with the graph of $\mathbf{D}$. It follows that $(\mathcal{L}_C)_j \subseteq (\mathcal{L}_D)_j$ for each j. □

Ignoring numerical cancellation, the edges in the graph of $\mathbf{A}\mathbf{A}^\mathsf{T}$ are a subset of the edges in the graph of $\mathbf{A}\mathbf{A}^\mathsf{T} + \mathbf{w}\mathbf{w}^\mathsf{T}$. By Proposition 3.2, we conclude that the edges in the graphs of the associated Cholesky factors satisfy the same inclusion. As a result, if the columns of $\mathbf{A}$ and the vectors $\mathbf{w}$ used in the update $\mathbf{A}\mathbf{A}^\mathsf{T} + \mathbf{w}\mathbf{w}^\mathsf{T}$ are all chosen from the columns of some fixed matrix $\mathbf{B}$, and if a fill-reducing permutation $\mathbf{P}$ can be found for which the Cholesky factors of $\mathbf{P}\mathbf{B}\mathbf{B}^\mathsf{T}\mathbf{P}^\mathsf{T}$ are sparse ([1] for example), then by Proposition 3.2, the Cholesky factors of $\mathbf{P}\mathbf{A}\mathbf{A}^\mathsf{T}\mathbf{P}^\mathsf{T}$ and of $\mathbf{P}(\mathbf{A}\mathbf{A}^\mathsf{T} + \mathbf{w}\mathbf{w}^\mathsf{T})\mathbf{P}^\mathsf{T}$ will be at least as sparse as those of $\mathbf{P}\mathbf{B}\mathbf{B}^\mathsf{T}\mathbf{P}^\mathsf{T}$.

4 Modifying the symbolic factors

Let $\overline{\mathbf{A}}$ be the modified version of $\mathbf{A}$. Again, we put a bar over a matrix or a set or a multiset to denote its value after the update or downdate is complete. In an update $\overline{\mathbf{A}}$ is obtained from $\mathbf{A}$ by appending the column $\mathbf{w}$ on the right, while in a downdate, $\overline{\mathbf{A}}$ is obtained from $\mathbf{A}$ by deleting the column $\mathbf{w}$ from $\mathbf{A}$. Hence, we have

$$\overline{\mathbf{A}\mathbf{A}}^\mathsf{T} = \mathbf{A}\mathbf{A}^\mathsf{T} + \sigma\mathbf{w}\mathbf{w}^\mathsf{T},$$

where σ is either $+1$ and $\mathbf{w}$ is the last column of $\overline{\mathbf{A}}$ (update) or σ is -1 and $\mathbf{w}$ is a column of $\mathbf{A}$ (downdate). Since $\overline{\mathbf{A}}$ and $\mathbf{A}$ differ by at most a single column, it follows from Proposition 3.2 that $\mathcal{L}_j \subseteq \overline{\mathcal{L}}_j$ for each j during an update, while $\overline{\mathcal{L}}_j \subseteq \mathcal{L}_j$ during a downdate. Moreover, the multisets associated with the Cholesky factor of either the updated or downdated matrix have the structure described in the following theorem:

Theorem 4.1 *Let k be the index associated with the first nonzero component of $\mathbf{w}$. For an update, $\mathcal{P}(k) \subseteq \overline{\mathcal{P}}(k)$ and $\overline{\mathcal{L}}_i^\sharp = \mathcal{L}_i^\sharp$ for all $i \in \overline{\mathcal{P}}(k)^c$, the complement of $\overline{\mathcal{P}}(k)$. That is, $\overline{\mathcal{L}}_i^\sharp = \mathcal{L}_i^\sharp$ for all i except when i is k or one of the new ancestors of k. For a downdate, $\overline{\mathcal{P}}(k) \subseteq \mathcal{P}(k)$ and $\overline{\mathcal{L}}_i^\sharp = \mathcal{L}_i^\sharp$ for all $i \in \mathcal{P}(k)^c$. That is, $\overline{\mathcal{L}}_i^\sharp = \mathcal{L}_i^\sharp$ for all i except when i is k or one of the old ancestors of k.*

Proof. To begin, let us consider an update. We will show that each element of $\mathcal{P}(k)$ is a member of $\overline{\mathcal{P}}(k)$ as well. Clearly, k lies in both $\mathcal{P}(k)$ and $\overline{\mathcal{P}}(k)$. Proceeding by induction, suppose that $\pi^0(k), \pi^1(k), \pi^2(k), \cdots, \pi^j(k) \in \overline{\mathcal{P}}(k)$, and define $l = \pi^j(k)$. We need to show that $\pi(l) = \pi(\pi^j(k)) = \pi^{j+1}(k) \in \overline{\mathcal{P}}(k)$ to complete the induction. Since $l \in \overline{\mathcal{P}}(k)$, we have $l = \overline{\pi}^h(k)$ for some h. If $\overline{\pi}(l) = \pi(l)$, then

$$\overline{\pi}^{h+1}(k) = \overline{\pi}(\overline{\pi}^h(k)) = \overline{\pi}(l) = \pi(l) = \pi(\pi^j(k)) = \pi^{j+1}(k).$$

Since $\pi^{j+1}(k) = \overline{\pi}^{h+1}(k) \in \overline{\mathcal{P}}(k)$, the induction step is complete, and $\pi^{j+1}(k) \in \overline{\mathcal{P}}(k)$.

If $\overline{\pi}(l) \neq \pi(l)$, then by Proposition 3.2, $\overline{\pi}(l) < \pi(l)$, and the following relation holds for $p = 1$:

$$\overline{\pi}^1(l) < \overline{\pi}^2(l) < \cdots < \overline{\pi}^p(l) < \pi(l). \tag{3}$$

Now suppose that (3) holds for some integer $p \geq 1$, and let q denote $\overline{\pi}^p(l)$. By Proposition 3.2, $\pi(l) \in \mathcal{L}_l \subseteq \overline{\mathcal{L}}_l$, and combining this with (3), $q = \overline{\pi}^p(l) < \pi(l) \in \overline{\mathcal{L}}_l$. It follows from (1) that $\pi(l) \in \overline{\mathcal{L}}_q$ for $q = \overline{\pi}^p(l)$. By the definition of the parent,

$$\overline{\pi}(q) = \overline{\pi}(\overline{\pi}^p(l)) = \overline{\pi}^{p+1}(l) \leq \pi(l) \in \overline{\mathcal{L}}_q.$$

Hence, either $\overline{\pi}^{p+1}(l) = \pi(l)$ or (3) holds with p replaced by $p+1$. Since (3) is violated for p sufficiently large, we conclude that there exists an integer p such that $\overline{\pi}^p(l) = \pi(l)$, from which it follows that

$$\pi^{j+1}(k) = \pi(\pi^j(k)) = \pi(l) = \overline{\pi}^p(l) = \overline{\pi}^p(\overline{\pi}^h(k)) = \overline{\pi}^{p+h}(k) \in \overline{\mathcal{P}}(k).$$

Since $\pi^{j+1}(k) \in \overline{\mathcal{P}}(k)$, the induction step is complete and $\mathcal{P}(k) \subseteq \overline{\mathcal{P}}(k)$.

Suppose that $l \in \overline{\mathcal{P}}(k)^c$. It is now important to recall that k is the index of the first nonzero component of $\mathbf{w}$, the vector appearing in the update. Observe that l cannot equal k since $l \in \overline{\mathcal{P}}(k)^c$ and $k \in \overline{\mathcal{P}}(k)$. The proof that $\overline{\mathcal{L}}_l^\sharp = \mathcal{L}_l^\sharp$ is by induction on the depth d defined by

$$d(l) = \max\{i : \pi^i(j) = l \text{ for some } j\}.$$

If $d(l) = 0$, then l has no children, and the child loop of Algorithm 2 will be skipped when either $\overline{\mathcal{L}}_l^\sharp$ or $\mathcal{L}_l^\sharp$ are evaluated. And since $l \neq k$, the pattern associated with $\mathbf{w}$ cannot be added into $\overline{\mathcal{L}}_l^\sharp$. Hence, when $d(l) = 0$, the identity $\overline{\mathcal{L}}_l^\sharp = \mathcal{L}_l^\sharp$ is trivial. Now, assuming that for some $p \geq 0$, we have $\overline{\mathcal{L}}_l^\sharp = \mathcal{L}_l^\sharp$ whenever $l \in \overline{\mathcal{P}}(k)^c$ and $d(l) \leq p$, let us suppose that $d(l) = p + 1$. If $i \in \pi^{-1}(l)$, then $d(i) \leq p$. Hence, $\overline{\mathcal{L}}_i^\sharp = \mathcal{L}_i^\sharp$ for $i \in \pi^{-1}(l)$ by the induction assumption. And since $l \neq k$, the pattern of $\mathbf{w}$ is not added to $\overline{\mathcal{L}}_l^\sharp$. Consequently, when Algorithm 2 is executed, we have $\overline{\mathcal{L}}_l^\sharp = \mathcal{L}_l^\sharp$, which completes the induction step.

Now consider the downdate part of the theorem. Rearranging the downdate relation $\overline{\mathbf{A}\mathbf{A}}^\mathsf{T} = \mathbf{A}\mathbf{A}^\mathsf{T} - \mathbf{w}\mathbf{w}^\mathsf{T}$, we have

$$\mathbf{A}\mathbf{A}^\mathsf{T} = \overline{\mathbf{A}\mathbf{A}}^\mathsf{T} + \mathbf{w}\mathbf{w}^\mathsf{T}.$$

Hence, in a downdate, we can think of $\mathbf{A}$ as the updated version of $\overline{\mathbf{A}}$. Consequently, the second part of the theorem follows directly from the first part. □

4.1 Symbolic update algorithm

We now present an algorithm for evaluating the new pattern $\overline{\mathcal{L}}$ associated with an update. Based on Theorem 4.1, the only sets $\overline{\mathcal{L}}_j$ that change are those associated with $\overline{\mathcal{P}}(k)$ where k is the index of the first nonzero component of $\mathbf{w}$. Referring to Algorithm 2, we can set $j = k$, $j = \overline{\pi}(k)$, $j = \overline{\pi}^2(k)$, and so on, marching up the path from j, and evaluate all the changes induced by the additional column in $\mathbf{A}$. In order to do the bookkeeping, there are at most four cases to consider:

Case 1. $j = k$: At the start of the new path, we need to add the pattern for $\mathbf{w}$ to $\mathcal{L}_j^\sharp$.

Case 2. $c \in \overline{\mathcal{P}}(k)$, $j \in \overline{\mathcal{P}}(k)$, $j > k$, and $c \in \overline{\pi}^{-1}(j) \cap \pi^{-1}(j)$: In this case, c is a child of j in both the new and the old elimination tree. Since the pattern $\overline{\mathcal{L}}_c$ may differ from $\mathcal{L}_c$, we need to add the difference to $\overline{\mathcal{L}}_j^\sharp$. Since j has a unique child on the path $\overline{\mathcal{P}}(k)$, there is at most one node c that satisfies these conditions. Also note that if

$$c \in \overline{\pi}^{-1}(j) \cap \pi^{-1}(j) \cap \overline{\mathcal{P}}(k)^c,$$

then by Theorem 4.1, $\overline{\mathcal{L}}_c = \mathcal{L}_c$ and hence, this node does not lead to an adjustment to $\overline{\mathcal{L}}_j^\sharp$ in Algorithm 2.

Case 3. $j \in \overline{\mathcal{P}}(k)$, $j > k$, and $c \in \overline{\pi}^{-1}(j) \setminus \pi^{-1}(j)$: In this case, c is a child of j in the new elimination tree, but not in the old tree, and the entire set $\overline{\mathcal{L}}_c$ should be added to $\overline{\mathcal{L}}_j^\sharp$ since it was not included in $\mathcal{L}_j^\sharp$. By Theorem 4.1, $\overline{\pi}(p) = \pi(p)$ for all $p \in \overline{\mathcal{P}}(k)^c$. Since $c \in \overline{\pi}^{-1}(j)$, but $c \notin \pi^{-1}(j)$, it follows that $\overline{\pi}(c) = j \neq \pi(c)$, and hence, $c \notin \overline{\mathcal{P}}(k)^c$, or equivalently, $c \in \overline{\mathcal{P}}(k)$. Again, since each node on the

path $\overline{\mathcal{P}}(k)$ from k has only one child on the path, there is at most one node c satisfying these conditions, and it lies on the path $\overline{\mathcal{P}}(k)$.

Case 4. $j \in \overline{\mathcal{P}}(k)$, $j > k$, and $c \in \pi^{-1}(j) \setminus \overline{\pi}^{-1}(j)$: In this case, c is a child of j in the old elimination tree, but not in the new tree, and the set $\mathcal{L}_c$ should be subtracted from $\overline{\mathcal{L}}_j^\sharp$ since it was previously added to $\mathcal{L}_j^\sharp$. Since $\pi(p) = \overline{\pi}(p)$ for each $p \in \overline{\mathcal{P}}(k)^c$, the fact that $\pi(c) = j \neq \overline{\pi}(c)$ implies that $c \in \overline{\mathcal{P}}(k)$. In the algorithm that follows, we refer to nodes c that satisfy these conditions as *lost children*.

In each of the cases above, every node c that led to adjustments in the pattern was located on the path $\overline{\mathcal{P}}(k)$. In the detailed algorithm that appears below, we simply march up the path from k to the root making the adjustments enumerated above.

Algorithm 3 (Symbolic update, add new column w)

Case 1: first node in the path
$k = \min \{i : w_i \neq 0\}$
$\overline{\mathcal{L}}_k^\sharp = \mathcal{L}_k^\sharp + \{i : w_i \neq 0\}$
$\overline{\pi}(k) = \min \overline{\mathcal{L}}_k \setminus \{k\}$
$c = k$
$j = \overline{\pi}(c)$
while $j \neq 0$ **do**
  if $j = \pi(c)$ **then**
    Case 2: c is an old child of j, possibly changed
    $\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp + (\overline{\mathcal{L}}_c \setminus \mathcal{L}_c)$
  else
    Case 3: c is a new child of j and a lost child of $\pi(c)$
    $\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp + (\overline{\mathcal{L}}_c \setminus \{c\})$
    place c in lost-child-queue of $\pi(c)$
  end if
  Case 4: consider each lost child of j
  for each c in lost-child-queue of j **do**
    $\overline{\mathcal{L}}_j^\sharp = \overline{\mathcal{L}}_j^\sharp - (\mathcal{L}_c \setminus \{c\})$
  end for
  $\overline{\pi}(j) = \min \overline{\mathcal{L}}_j \setminus \{j\}$
  $c = j$
  $j = \overline{\pi}(c)$
end while
$\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp$ and $\overline{\pi}(j) = \pi(j)$ for all $j \in \overline{\mathcal{P}}(k)^c$
end Algorithm 3

The time taken by this algorithm is given by the following lemma.

Lemma 4.2 *The time to execute Algorithm 3 is bounded above by a constant times the number of entries associated with patterns for nodes on the new path $\overline{\mathcal{P}}(k)$. That is, the time is*

$$O\left(\sum_{j\in\overline{\mathcal{P}}(k)} |\overline{\mathcal{L}}_j|\right).$$

Proof. In Algorithm 3, we simply march up the path $\overline{\mathcal{P}}(k)$ making adjustments to $\overline{\mathcal{L}}_j^\sharp$ as we proceed. At each node j, we take time proportional to $|\overline{\mathcal{L}}_j|$, plus the time taken to process the children of j. Each node is visited as a child c at most twice, since it falls into one or two of the four cases enumerated above (a node c can be a new child of one node, and a lost child of another). If this work (proportional to $|\mathcal{L}_c|$ or $|\overline{\mathcal{L}}_c|$) is accounted to step c instead of j, the time to make the adjustment to the pattern is bounded above by a constant times either $|\mathcal{L}_j|$ or $|\overline{\mathcal{L}}_j|$. Since $|\mathcal{L}_j| \le |\overline{\mathcal{L}}_j|$ by Theorem 4.1, the proof is complete. □

In practice, we can reduce the execution time for Algorithm 3 by skipping over any nodes for which $\mathcal{L}_j^\sharp = \overline{\mathcal{L}}_j^\sharp$. That is, if the current node j has no lost children, and if its child c falls under case 2 with $\overline{\mathcal{L}}_c = \mathcal{L}_c$, then we have $\mathcal{L}_j^\sharp = \overline{\mathcal{L}}_j^\sharp$ and the update can be skipped. The execution time for this modified algorithm, which is the one we implement, is

$$O\left(|\overline{\mathcal{P}}(k)| + \sum_{j\in\overline{\mathcal{P}}(k)\wedge\mathcal{L}_j^\sharp\neq\overline{\mathcal{L}}_j^\sharp} |\overline{\mathcal{L}}_j|\right).$$

4.2 Symbolic downdate algorithm

Let us consider the removal of a column $\mathbf{w}$ from $\mathbf{A}$ and let k be the index of the first nonzero entry in $\mathbf{w}$. The symbolic downdate algorithm is analogous to the symbolic update algorithm, but the roles of $\mathcal{P}(k)$ and $\overline{\mathcal{P}}(k)$ are interchanged in accordance with Theorem 4.1. Instead of adding entries to $\mathcal{L}_j^\sharp$, we subtract entries, instead of lost child queues, we have new child queues, instead of walking up the path $\overline{\mathcal{P}}(k)$, we walk up the path $\mathcal{P}(k) \supseteq \overline{\mathcal{P}}(k)$.

Algorithm 4 (Symbolic downdate, remove column w)

Case 1: first node in the path
$k = \min\{i : w_i \neq 0\}$
$\overline{\mathcal{L}}_k^\sharp = \mathcal{L}_k^\sharp - \{i : w_i \neq 0\}$
$\overline{\pi}(k) = \min \overline{\mathcal{L}}_k \setminus \{k\}$

$c = k$
$j = \pi(c)$
while $j \neq 0$ **do**
  if $j = \overline{\pi}(c)$ **then**
    Case 2: c is an old child of j, possibly changed
    $\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp - (\mathcal{L}_c \setminus \overline{\mathcal{L}}_c)$
  else
    Case 3: c is a lost child of j and a new child of $\overline{\pi}(c)$
    $\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp - (\mathcal{L}_c \setminus \{c\})$
    place c in new-child-queue of $\overline{\pi}(c)$
  end if
  Case 4: consider each new child of j
  for each c in new-child-queue of j **do**
    $\overline{\mathcal{L}}_j^\sharp = \overline{\mathcal{L}}_j^\sharp + (\overline{\mathcal{L}}_c \setminus \{c\})$
  end for
  $\overline{\pi}(j) = \min\ \overline{\mathcal{L}}_j \setminus \{j\}$
  $c = j$
  $j = \pi(c)$
end while
$\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp$ and $\overline{\pi}(j) = \pi(j)$ for all $j \in \mathcal{P}(k)^c$
end Algorithm 4

Similar to Algorithm 3, the execution time obeys the following estimate:

Lemma 4.3 *The time to execute Algorithm 4 is bounded above by a constant times the number of entries associated with patterns for nodes on the old path* $\mathcal{P}(k)$*. That is, the time is*

$$O\left(\sum_{j \in \mathcal{P}(k)} |\mathcal{L}_j|\right).$$

Again, we achieve in practice some speedup when we check whether or not $\mathcal{L}_j^\sharp$ changes for any given node $j \in \mathcal{P}(k)$. The time taken by this modified Algorithm 4 is

$$O\left(|\mathcal{P}(k)| + \sum_{j \in \mathcal{P}(k) \wedge \mathcal{L}_j^\sharp \neq \overline{\mathcal{L}}_j^\sharp} |\mathcal{L}_j|\right).$$

5 The numerical factors

When we add or delete a column in $\mathbf{A}$, we update or downdate the symbolic factorization in order to determine the location in the Cholesky factor of either new

nonzero entries or nonzero entries that are now zero. Knowing the location of the nonzero entries, we can update the numerical value of these entries. We first consider the case when $\mathbf{A}$ and $\mathbf{L}$ are dense and draw on the ideas of [15]. Then we show how the method extends to the sparse case.

5.1 Dense matrices

Our algorithm to implement the numerical update and downdate is based on the Method C5 in [15] for dense matrices. To summarize their approach, we start by writing

$$\overline{\mathbf{A}\mathbf{A}}^{\mathsf{T}} = \mathbf{A}\mathbf{A}^{\mathsf{T}} + \sigma\mathbf{w}\mathbf{w}^{\mathsf{T}} = \mathbf{L}\mathbf{L}^{\mathsf{T}} + \sigma\mathbf{w}\mathbf{w}^{\mathsf{T}} = \mathbf{L}(\mathbf{I} + \sigma\mathbf{v}\mathbf{v}^{\mathsf{T}})\mathbf{L}^{\mathsf{T}}, \tag{4}$$

where $\mathbf{v} = \mathbf{L}^{-1}\mathbf{w}$. Observe that

$$\mathbf{I} + \sigma\mathbf{v}\mathbf{v}^{\mathsf{T}} = (\mathbf{I} + \tau\mathbf{v}\mathbf{v}^{\mathsf{T}})^2, \tag{5}$$

where $\tau = \sigma/(1 + \sqrt{1 + \sigma\|\mathbf{v}\|^2})$. The quantity $1 + \sigma\|\mathbf{v}\|^2$ is positive on account of our assumption that both $\mathbf{A}\mathbf{A}^{\mathsf{T}}$ and $\overline{\mathbf{A}\mathbf{A}}^{\mathsf{T}}$ are positive definite. That is, since $\overline{\mathbf{A}\mathbf{A}}^{\mathsf{T}} = \mathbf{L}(\mathbf{I} + \sigma\mathbf{v}\mathbf{v}^{\mathsf{T}})\mathbf{L}^{\mathsf{T}}$, the matrices $\overline{\mathbf{A}\mathbf{A}}^{\mathsf{T}}$ and $\mathbf{I} + \sigma\mathbf{v}\mathbf{v}^{\mathsf{T}}$ are congruent, and by Sylvester's law of inertia (see [24]), they have the same number of positive, negative, and zero eigenvalues. The eigenvalues of $\mathbf{I} + \sigma\mathbf{v}\mathbf{v}^{\mathsf{T}}$ are one, with multiplicity $m - 1$ (corresponding to the eigenvectors orthogonal to $\mathbf{v}$) and $1 + \sigma\|\mathbf{v}\|^2$ (corresponding to the eigenvector $\mathbf{v}$). Since $\overline{\mathbf{A}\mathbf{A}}^{\mathsf{T}}$ is positive definite, its eigenvalues are all positive, which implies that $1 + \sigma\|\mathbf{v}\|^2 > 0$.

Combining (4) and (5), we have

$$\overline{\mathbf{A}\mathbf{A}}^{\mathsf{T}} = \mathbf{L}(\mathbf{I} + \tau\mathbf{v}\mathbf{v}^{\mathsf{T}})(\mathbf{I} + \tau\mathbf{v}\mathbf{v}^{\mathsf{T}})\mathbf{L}^{\mathsf{T}}.$$

A sequence of Givens rotations, the product being denoted $\mathbf{G}_1$, is chosen so that $\mathbf{v}^{\mathsf{T}}\mathbf{G}_1 = \|\mathbf{v}\|\mathbf{e}_1^{\mathsf{T}}$ where $\mathbf{e}_1$ is the vector with every entry zero except for the first entry. The matrix $(\mathbf{I} + \tau\mathbf{v}\mathbf{v}^{\mathsf{T}})\mathbf{G}_1$ has a lower Hessenberg structure. A second sequence of Givens rotations, the product being denoted $\mathbf{G}_2$, is chosen so that $(\mathbf{I} + \tau\mathbf{v}\mathbf{v}^{\mathsf{T}})\mathbf{G}_1\mathbf{G}_2$ is a lower triangular matrix, denoted $\mathbf{G}$, of the following form:

$$\mathbf{G} = \begin{bmatrix} \delta_1 & & & & \\ v_2\gamma_1 & \delta_2 & & & \\ v_3\gamma_1 & v_3\gamma_2 & \delta_3 & & \\ \vdots & \vdots & \vdots & \ddots & \\ v_m\gamma_1 & v_m\gamma_2 & v_m\gamma_3 & \dots & \delta_m \end{bmatrix}$$

where δ and γ are vectors computed in Algorithm 5 below. The diagonal elements of $\mathbf{G}$ are given by $g_{jj} = \delta_j$, while the elements below the diagonal are $g_{ij} = v_i\gamma_j$.

Algorithm 5 (Compute G, dense case)

$\mathbf{v} = \mathbf{L}^{-1}\mathbf{w}$
$\omega = \max\{i : v_i \neq 0\}$
$\rho = \|\mathbf{v}\|_2$
if $\omega = 1$ **then**
$\delta_1 = \sqrt{1 + \sigma\rho^2}$
else
$t = v_\omega$
find $\mathbf{G}_1$*, to zero out all but first entry of* $\mathbf{v}$*:*
for $j = \omega - 1$ **down to** 1 **do**
$s = \sqrt{t^2 + v_j^2}$
$\beta_j = v_j/(st)$
$s_j = t/s$
$t = s$
end for
$\eta = 1/\rho + \sigma\rho/(1 + \sqrt{1 + \sigma\rho^2})$
find $\mathbf{G}_2$*, and combine to obtain* $\mathbf{G}$*:*
for $j = 1$ **to** $\omega - 1$ **do**
$\delta_j = \sqrt{(\eta v_j)^2 + s_j^2}$
$c = (\eta v_j)/\delta_j$
$s = -s_j/\delta_j$
$\gamma_j = c\eta + s\beta_j$
$\eta = c\beta_j - s\eta$
end for
$\delta_\omega = v_\omega\eta$
end if
$\delta_i = 1$ for $\omega < i \leq m$
end Algorithm 5

In Algorithm 6 below, we evaluate the new Cholesky factor $\overline{\mathbf{L}} = \mathbf{LG}$, without forming $\mathbf{G}$ explicitly [15] by taking advantage of the special structure of $\mathbf{G}$. The product is computed column by column, moving from right to left. In practice, $\mathbf{L}$ can be overwritten with $\overline{\mathbf{L}}$.

Algorithm 6 (Compute $\overline{\mathbf{L}} = \mathbf{LG}$, dense case)

$t_{1\ldots n} = 0$
for $j = \omega$ **down to** 1 **do**
for $i = j$ **to** m **do**
$\overline{l}_{ij} = \delta_j l_{ij} + t_i\gamma_j$

$t_i = t_i + v_j l_{ij}$
end for
end for
end Algorithm 6

Observe that about 1/3 of the multiplies can be eliminated if $\mathbf{L}$ is stored as a product $\tilde{\mathbf{L}}\mathbf{D}$, where $\mathbf{D}$ is a diagonal matrix. The components of δ can be absorbed into $\mathbf{D}$, with γ adjusted accordingly, and δ in Algorithm 6 can be eliminated. We did not exploit this simplification for the numerical results reported in §8.

5.2 The sparsity pattern of v

In the sparse case, $\mathbf{v} = \mathbf{L}^{-1}\mathbf{w}$ is sparse, and its nonzero pattern is crucial. Since the elements of $\mathbf{G}$ satisfy $g_{ij} = v_i\gamma_j$ for each $i > j$, we conclude that only the columns of $\mathbf{L}$ associated with the nonzero components of $\mathbf{v}$ enter into the computation of $\overline{\mathbf{L}}$. The nonzero pattern of $\mathbf{v}$ can be found using the following lemma.

Lemma 5.1 *The nodes reachable from any given node k by path(s) in the directed graph $G(\mathbf{L}^\mathsf{T})$ coincide with the path $\mathcal{P}(k)$.*

Proof. If $\mathcal{P}(k)$ has a single element, the lemma holds. Proceeding by induction, suppose that the lemma holds for all k for which $|\mathcal{P}(k)| \leq j$. Now, if $\mathcal{P}(k)$ has $j+1$ elements, then by the induction hypothesis, the nodes reachable from $\pi(k)$ by path(s) in the directed graph $G(\mathbf{L}^\mathsf{T})$ coincide with the path $\mathcal{P}(\pi(k))$. The nodes reachable in one step from k consist of the elements of $\mathcal{L}_k$. By Proposition 3.1, each of the elements of $\mathcal{L}_k$ is contained in the path $\mathcal{P}(k)$. If $i \in \mathcal{L}_k$, $i \neq k$, then $|\mathcal{P}(i)| \leq j$. By the induction hypothesis, the nodes reachable from i coincide with $\mathcal{P}(i) \subseteq \mathcal{P}(k)$. The nodes reachable from k consist of $\{k\}$ union the nodes reachable from $\mathcal{L}_k$. Since $k \in \mathcal{P}(k)$, it follows that the nodes reachable from k are contained in $\mathcal{P}(k)$. On the other hand, for each p, the element of $\mathbf{L}^\mathsf{T}$ in row $\pi^p(k)$ and column $\pi^{p+1}(k)$ is nonzero. Hence, all the elements of $\mathcal{P}(k)$ are reachable from k. Since the nodes in $\mathcal{P}(k)$ coincide with the the nodes reachable from k by path(s) in the directed graph $G(\mathbf{L}^\mathsf{T})$, the induction step is complete. □

Theorem 5.2 *During symbolic downdate $\overline{\mathbf{A}\mathbf{A}}^\mathsf{T} = \mathbf{A}\mathbf{A}^\mathsf{T} - \mathbf{w}\mathbf{w}^\mathsf{T}$ (where $\mathbf{w}$ is a column of $\mathbf{A}$), the nonzero pattern of $\mathbf{v} = \mathbf{L}^{-1}\mathbf{w}$ is equal to the path $\mathcal{P}(k)$ in the (old) elimination tree of $\mathbf{L}$ where*

$$k = \min\{i : w_i \neq 0\}. \tag{6}$$

Proof. Let $\mathcal{W} = \{i : w_i \neq 0\}$. Theorem 5.1 of Gilbert [10, 11, 14] states that the nonzero pattern of $\mathbf{v}$ is the set of nodes reachable from the nodes in $\mathcal{W}$ by paths

in the directed graph $G(\mathbf{L}^\mathsf{T})$. By Algorithm 1, $\mathcal{W} \subseteq \mathcal{L}_k$. Hence, each element of $\mathcal{W}$ is reachable from k by a path of length one, and the nodes reachable from $\mathcal{W}$ are a subset of the nodes reachable from k. Conversely, since $k \in \mathcal{W}$, the nodes reachable from k are a subset of the nodes reachable from $\mathcal{W}$. Combining these inclusions, the nodes reachable from k and from $\mathcal{W}$ are the same, and by Lemma 5.1, the nodes reachable from k coincide with the path $\mathcal{P}(k)$. □

Corollary 5.3 *During symbolic update* $\overline{\mathbf{AA}}^\mathsf{T} = \mathbf{AA}^\mathsf{T} + \mathbf{ww}^\mathsf{T}$, *the nonzero pattern of* $\mathbf{v} = \mathbf{L}^{-1}\mathbf{w}$ *is equal to the path* $\overline{\mathcal{P}}(k)$ *in the (new) elimination tree of* $\overline{\mathbf{L}}$ *where* k *is defined in (6).*

Proof. Since $\mathbf{LL}^\mathsf{T} = \overline{\mathbf{AA}}^\mathsf{T} - \mathbf{ww}^\mathsf{T}$, we can view $\mathbf{L}$ as the Cholesky factor for the downdate $\overline{\mathbf{AA}}^\mathsf{T} - \mathbf{ww}^\mathsf{T}$. Hence, we can apply Theorem 5.2, in effect replacing $\mathcal{P}$ by $\overline{\mathcal{P}}$. □

5.3 Sparse matrices

In the dense algorithm presented at the start of this section, we write

$$\overline{\mathbf{AA}}^\mathsf{T} = \mathbf{L}(\mathbf{I} + \tau\mathbf{vv}^\mathsf{T})(\mathbf{I} + \tau\mathbf{vv}^\mathsf{T})\mathbf{L}^\mathsf{T} \tag{7}$$

where $\mathbf{v} = \mathbf{L}^{-1}\mathbf{w}$. To be specific, let us consider the case where (7) corresponds to an update. The nonzero elements of $\mathbf{I} + \tau\mathbf{vv}^\mathsf{T}$ either lie along the diagonal or at the intersection of rows $i \in \overline{\mathcal{P}}(k)$ and columns $j \in \overline{\mathcal{P}}(k)$, where k is defined in (6). In essence, we can extract the submatrix of $\mathbf{I} + \tau\mathbf{vv}^\mathsf{T}$ corresponding to these rows and columns, we can apply Algorithm 5 to this dense submatrix, we can modify the submatrix of $\mathbf{L}$ consisting of rows and columns associated with $\overline{\mathcal{P}}(k)$, and then we can reinsert this dense submatrix in the m by m sparse matrix $\mathbf{L}$. At the algebraic level, we can think of this process in the following way: If $\mathbf{P}$ is the permutation matrix with the property that the columns associated with $\overline{\mathcal{P}}(k)$ are moved to the first $|\overline{\mathcal{P}}(k)|$ columns by multiplication on the right, then we have

$$\overline{\mathbf{AA}}^\mathsf{T} = \mathbf{P}(\mathbf{P}^\mathsf{T}\mathbf{LP})(\mathbf{P}^\mathsf{T}(\mathbf{I} + \tau\mathbf{vv}^\mathsf{T})\mathbf{P})(\mathbf{P}^\mathsf{T}(\mathbf{I} + \tau\mathbf{vv}^\mathsf{T})\mathbf{P})(\mathbf{P}^\mathsf{T}\mathbf{L}^\mathsf{T}\mathbf{P})\mathbf{P}^\mathsf{T}.$$

The matrix $\mathbf{P}^\mathsf{T}(\mathbf{I} + \tau\mathbf{vv}^\mathsf{T})\mathbf{P}$ is a 2 by 2 block diagonal with the identity in the lower right corner and a dense upper left corner $\mathbf{V}$,

$$\begin{aligned}\left(\mathbf{P}^\mathsf{T}\mathbf{LP}\right)\left(\mathbf{P}^\mathsf{T}(\mathbf{I} + \tau\mathbf{vv}^\mathsf{T})\mathbf{P}\right) &= \begin{pmatrix} \mathbf{L}_{11} & \mathbf{L}_{12} \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{pmatrix}\begin{pmatrix} \mathbf{V} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{L}_{11}\mathbf{V} & \mathbf{L}_{12} \\ \mathbf{0} & \mathbf{L}_{22} \end{pmatrix}. \end{aligned} \tag{8}$$

The elements of $\mathbf{L}_{21}$ correspond to the elements l_{ij} of $\mathbf{L}$ for which $i \in \overline{\mathcal{P}}(k)^c$ and $j \in \overline{\mathcal{P}}(k)$. By Proposition 3.1, $\mathcal{L}_j \subseteq \overline{\mathcal{L}}_j \subseteq \overline{\mathcal{P}}(k)$ when $j \in \overline{\mathcal{P}}(k)$. Since $\overline{\mathcal{P}}(k)^c \subseteq \overline{\mathcal{L}}_j^c$, it follows that $i \in \overline{\mathcal{L}}_j^c$ when $i \in \overline{\mathcal{P}}(k)^c$. Hence, $l_{ij} = 0$ and $\mathbf{L}_{21} = \mathbf{0}$. Note that in general $\mathbf{L}_{12} \neq \mathbf{0}$, which is why Algorithm 6 is column-oriented. When (8) is multiplied by the Givens rotations computed by Algorithm 5, we obtain

$$\begin{aligned}\left(\mathbf{P}^{\mathsf{T}}\overline{\mathbf{L}}\mathbf{P}\right) &= \begin{pmatrix} \mathbf{L}_{11}\mathbf{V} & \mathbf{L}_{12} \\ \mathbf{0} & \mathbf{L}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{G}_1\mathbf{G}_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{L}_{11}\mathbf{G} & \mathbf{L}_{12} \\ \mathbf{0} & \mathbf{L}_{22} \end{pmatrix} \\ &= \begin{pmatrix} \overline{\mathbf{L}}_{11} & \mathbf{L}_{12} \\ \mathbf{0} & \mathbf{L}_{22} \end{pmatrix}. \end{aligned} \tag{9}$$

We apply $\mathbf{P}$ to the left and $\mathbf{P}^{\mathsf{T}}$ to the right in (9) to obtain $\overline{\mathbf{L}}$. Neither $\mathbf{L}_{12}$ nor $\mathbf{L}_{22}$ are modified.

When (7) corresponds to a downdate, the discussion is the same as for an update except that $\mathcal{P}(k)$ replaces $\overline{\mathcal{P}}(k)$, and the role of $\mathcal{L}_j$ and $\overline{\mathcal{L}}_j$ are interchanged. In summary, the sparse update or downdate of a Cholesky factorization can be accomplished by first evaluating $\mathbf{v} = \mathbf{L}^{-1}\mathbf{w}$ whose nonzero entries are contained in the path $\overline{\mathcal{P}}(k)$ or $\mathcal{P}(k)$ where k is defined in (6). We apply the dense Algorithm 5 to the vector of (symbolically) nonzero entries of $\mathbf{v}$. Then we update the entries in $\mathbf{L}$ in the rows and columns associated with indices in $\overline{\mathcal{P}}(k)$ or $\mathcal{P}(k)$ using Algorithm 6.

As the following result indicates, Algorithm 6 does not have to be applied to all of $\mathbf{L}_{11}$ because of its specific structure.

Proposition 5.4 *During symbolic downdate, $\mathbf{L}_{11}$ is a lower triangular matrix with a dense profile. That is, for each row $i > 1$, there is an integer $p_i < i$ such that $[\mathbf{L}_{11}]_{ij} = 0$ for all $1 \leq j < p_i$, and $[\mathbf{L}_{11}]_{ij} \neq 0$ for all $p_i \leq j \leq i$.*

Proof. The rows and columns columns of $\mathbf{L}_{11}$ correspond to the nodes in the path $\mathcal{P}(k)$. We assume that the nodes have been relabeled so that

$$\mathcal{P}(k) = \{1, 2, \cdots, |\mathcal{P}(k)|\}.$$

It follows that $[\mathbf{L}_{11}]_{i,i-1} \neq 0$ for all $i > 1$. If $[\mathbf{L}_{11}]_{ip} \neq 0$ for some p, then $i \in \mathcal{L}_p$, and by (1), if $j \in \mathcal{P}(p)$ and $j \leq i$, then $i \in \mathcal{L}_j$. Consequently, $[\mathbf{L}_{11}]_{ij} \neq 0$ for all $p \leq j \leq i$. Letting p_i be the smallest such index p, the proof is complete. □

Corollary 5.5 *During symbolic update $\overline{\mathbf{L}}_{11}$ is a lower triangular matrix with a dense profile. That is, for each row $i > 1$, there is an integer $p_i < i$ such that $[\overline{\mathbf{L}}_{11}]_{ij} = 0$ for all $1 \leq j < p_i$, and $[\overline{\mathbf{L}}_{11}]_{ij} \neq 0$ for all $p_i \leq j \leq i$.*

Proof. This follows immediately from Proposition 5.4, replacing $\mathcal{P}(k)$ with $\overline{\mathcal{P}}(k)$. □

As a consequence of Proposition 5.4, Corollary 5.5, and Proposition 3.2, we can skip over any index i in Algorithm 6 for which $j < p_i$. When Algorithm 6 is applied to the sparse $\mathbf{L}$, as opposed to the dense submatrix, the indices j and i take values in $\mathcal{P}(k)$ and $\mathcal{L}_j$ respectively for a downdate, while they take values in $\overline{\mathcal{P}}(k)$ and $\overline{\mathcal{L}}_j$, respectively, for an update.

6 Arbitrary symbolic and numerical factors

The methods we have developed for computing the modification to the Cholesky factors of $\mathbf{A}\mathbf{A}^\mathsf{T}$ corresponding to the addition or deletion of columns in $\mathbf{A}$ can be used to determine the effect on the Cholesky factors of a general symmetric positive definite matrix $\mathbf{M}$ of any symmetric change of the form $\mathbf{M} + \sigma\mathbf{w}\mathbf{w}^\mathsf{T}$ that preserves positive definiteness. We briefly describe how Algorithms 1 through 6 are modified for the general case.

Let $\mathcal{M}_j$ denote the nonzero pattern of the *lower triangular* part of $\mathbf{M}$:

$$\mathcal{M}_j = \{i \ : \ m_{ij} \neq 0 \text{ and } i \leq j\}.$$

The symbolic factorization of $\mathbf{M}$ [5, 6, 7, 8, 23] is obtained by replacing the union of $\mathcal{A}_k$ terms in Algorithm 1 with the set $\mathcal{M}_j$. With this change, $\mathcal{L}_j$ of Algorithm 1 is given by

$$\mathcal{L}_j = \{j\} \cup \left(\bigcup_{c \in \pi^{-1}(j)} \mathcal{L}_c \setminus \{c\} \right) \cup \mathcal{M}_j.$$

This leads to a change in Algorithm 2 for computing the multiplicities. The multiplicity of an index i in $\mathcal{L}_j$ becomes

$$m(i,j) = |\{k \in \pi^{-1}(j) : i \in \mathcal{L}_k\}| + (1 \text{ if } i \in \mathcal{M}_j, \text{ or } 0 \text{ otherwise}).$$

The for loop involving the $\mathcal{A}_k$ terms in Algorithm 2 is replaced by the single statement $\mathcal{L}_j^\sharp = \mathcal{L}_j^\sharp + \mathcal{M}_j$. More precisely, we have:

$$\left.\begin{array}{l} \textbf{for} \text{ each } k \text{ where } \min \mathcal{A}_k = j \textbf{ do} \\ \quad \mathcal{L}_j^\sharp = \mathcal{L}_j^\sharp + \mathcal{A}_k \\ \textbf{end for} \end{array}\right\} \quad \Rightarrow \quad \mathcal{L}_j^\sharp = \mathcal{L}_j^\sharp + \mathcal{M}_j$$

Entries are removed or added symbolically from $\mathbf{A}\mathbf{A}^\mathsf{T}$ by the deletion or addition of columns of $\mathbf{A}$, and numerical cancellation is ignored. Numerical cancellation of entries in $\mathbf{M}$ should not be ignored, however, because this is the only way that entries can be dropped from $\mathbf{M}$. When numerical cancellation is taken into account, neither

of the inclusions $\mathcal{M}_j \subseteq \overline{\mathcal{M}}_j$ or $\overline{\mathcal{M}}_j \subseteq \mathcal{M}_j$ may hold. We resolve this problem by using a symbolic modification scheme with two steps: a symbolic update phase in which new nonzero entries in $\mathbf{M} + \sigma\mathbf{w}\mathbf{w}^\mathsf{T}$ are taken into account, followed by a separate symbolic downdate phase to handle entries that become numerically zero. Since each modification step now involves an update phase followed by a downdate phase, we attach (in this section) a overbar to quantities associated with the update and an underbar to quantities associated with the downdate.

Let $\mathcal{W}$ be the nonzero pattern of $\mathbf{w}$, namely $\mathcal{W} = \{i : w_i \neq 0\}$. In the first symbolic phase, entries from $\mathcal{W}$ are symbolically added to $\mathcal{M}_j$ for each $j \in \mathcal{W}$. That is, if $i \notin \mathcal{M}_j$, but $i, j \in \mathcal{W}$ with $i > j$, then we add i to $\mathcal{M}_j$:

$$\overline{\mathcal{M}}_j = \mathcal{M}_j \cup \{i \in \mathcal{W} : i > j\}.$$

In the second symbolic phase, entries from $\mathcal{W}$ are symbolically deleted for each $j \in \mathcal{W}$:

$$\underline{\mathcal{M}}_j = \overline{\mathcal{M}}_j \setminus \{i \in \mathcal{W} : i > j, \;\; m_{ij} + \sigma w_i w_j = 0\}. \tag{10}$$

In practice, we need to introduce a drop tolerance t and replace the equality

$$m_{ij} + \sigma w_i w_j = 0$$

in (10) by the inequality $|m_{ij} + \sigma w_i w_j| \leq t$. For a general matrix, the analogue of Theorem 4.1 is the following:

Theorem 6.1 *If α is the first index for which $\mathcal{M}_\alpha \neq \overline{\mathcal{M}}_\alpha$, then $\mathcal{P}(\alpha) \subseteq \overline{\mathcal{P}}(\alpha)$ and $\overline{\mathcal{L}}_i^\sharp = \mathcal{L}_i^\sharp$ for all $i \in \overline{\mathcal{P}}(\alpha)^c$. If β is the first index for which $\overline{\mathcal{M}}_\beta \neq \underline{\mathcal{M}}_\beta$, then $\underline{\mathcal{P}}(\beta) \subseteq \overline{\mathcal{P}}(\beta)$ and $\underline{\mathcal{L}}_i^\sharp = \overline{\mathcal{L}}_i^\sharp$ for all $i \in \overline{\mathcal{P}}(\beta)^c$.*

In evaluating the modification in the symbolic factorization associated with $\mathbf{M} + \sigma\mathbf{w}\mathbf{w}^\mathsf{T}$, we start at the first index α where $\mathcal{M}_\alpha \neq \overline{\mathcal{M}}_\alpha$ and we march up the path $\overline{\mathcal{P}}(\alpha)$ making changes to $\mathcal{L}_j^\sharp$, obtaining $\overline{\mathcal{L}}_j^\sharp$. In the second phase, we start at the first index where $\overline{\mathcal{M}}_\beta \neq \underline{\mathcal{M}}_\beta$, and we march up the path $\overline{\mathcal{P}}(\beta)$ making changes to $\overline{\mathcal{L}}_j^\sharp$, obtaining $\underline{\mathcal{L}}_j^\sharp$. The analogue of Algorithm 3 in the general case only differs in the starting index (now α) and in the addition of the sets $\overline{\mathcal{M}}_j \setminus \mathcal{M}_j$ in each pass through the j-loop:

Algorithm 7a (Symbolic update phase, general matrix)

Case 1: first node in the path
$\alpha = \min \{i : \mathcal{M}_i \neq \overline{\mathcal{M}}_i\}$
$\overline{\mathcal{L}}_\alpha^\sharp = \mathcal{L}_\alpha^\sharp + \overline{\mathcal{M}}_\alpha \setminus \mathcal{M}_\alpha$
$\overline{\pi}(\alpha) = \min \; \overline{\mathcal{L}}_\alpha \setminus \{\alpha\}$

$c = \alpha$
$j = \overline{\pi}(c)$
while $j \neq 0$ **do**
$\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp + \overline{\mathcal{M}}_j \setminus \mathcal{M}_j$
if $j = \pi(c)$ **then**
Case 2: c is an old child of j, possibly changed
$\overline{\mathcal{L}}_j^\sharp = \overline{\mathcal{L}}_j^\sharp + (\overline{\mathcal{L}}_c \setminus \mathcal{L}_c)$
else
Case 3: c is a new child of j and a lost child of $\pi(c)$
$\overline{\mathcal{L}}_j^\sharp = \overline{\mathcal{L}}_j^\sharp + (\overline{\mathcal{L}}_c \setminus \{c\})$
place c in lost-child-queue of $\pi(c)$
end if
Case 4: consider each lost child of j
for each c in lost-child-queue of j **do**
$\overline{\mathcal{L}}_j^\sharp = \overline{\mathcal{L}}_j^\sharp - (\mathcal{L}_c \setminus \{c\})$
end for
$\overline{\pi}(j) = \min\ \overline{\mathcal{L}}_j \setminus \{j\}$
$c = j$
$j = \overline{\pi}(c)$
end while
$\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp$ and $\overline{\pi}(j) = \pi(j)$ for all $j \in \overline{\mathcal{P}}(\alpha)^c$
end Algorithm 7a

Similarly, the analogue of Algorithm 4 in the general case only differs in the starting index (now β), in the subtraction of the sets and $\overline{\mathcal{M}}_j \setminus \underline{\mathcal{M}}_j$ in each pass through the j-loop.

Algorithm 7b (Symbolic downdate phase, general matrix)
Case 1: first node in the path
$\beta = \min\{i : \overline{\mathcal{M}}_i \neq \underline{\mathcal{M}}_i\}$
$\underline{\mathcal{L}}_\beta^\sharp = \overline{\mathcal{L}}_\beta^\sharp - \overline{\mathcal{M}}_\beta \setminus \underline{\mathcal{M}}_\beta$
$\underline{\pi}(\beta) = \min\ \underline{\mathcal{L}}_\beta \setminus \{\beta\}$
$c = \beta$
$j = \overline{\pi}(c)$
while $j \neq 0$ **do**
$\underline{\mathcal{L}}_j^\sharp = \overline{\mathcal{L}}_j^\sharp - \overline{\mathcal{M}}_j \setminus \underline{\mathcal{M}}_j$
if $j = \underline{\pi}(c)$ **then**
Case 2: c is an old child of j, possibly changed
$\underline{\mathcal{L}}_j^\sharp = \underline{\mathcal{L}}_j^\sharp - (\overline{\mathcal{L}}_c \setminus \underline{\mathcal{L}}_c)$

else

Case 3: c is a lost child of j and a new child of $\underline{\pi}(c)$

$\underline{\mathcal{L}}_j^\sharp = \underline{\mathcal{L}}_j^\sharp - (\overline{\mathcal{L}}_c \setminus \{c\})$

place c in new-child-queue of $\underline{\pi}(c)$

end if

Case 4: consider each new child of j

for each c in new-child-queue of j **do**

$\underline{\mathcal{L}}_j^\sharp = \underline{\mathcal{L}}_j^\sharp + (\underline{\mathcal{L}}_c \setminus \{c\})$

end for

$\underline{\pi}(j) = \min\ \underline{\mathcal{L}}_j \setminus \{j\}$

$c = j$

$j = \overline{\pi}(c)$

end while

$\underline{\mathcal{L}}_j^\sharp = \overline{\mathcal{L}}_j^\sharp$ and $\underline{\pi}(j) = \overline{\pi}(j)$ for all $j \in \overline{\mathcal{P}}(\beta)^c$

end Algorithm 7b

Algorithms 5 and 6 are completely unchanged in the general case. They can be applied after the completion of Algorithm 7b so that we know the location of new nonzero entries in the Cholesky factor. They process the submatrix associated with rows and columns in $\overline{\mathcal{P}}(k)$ where k is the index of the first nonzero element of $\mathbf{w}$. When $\mathbf{M}$ has the form $\mathbf{A}\mathbf{A}^\mathsf{T}$ and when $\overline{\mathbf{M}}$ is gotten by either adding or deleting a column in $\mathbf{A}$, then assuming no numerical cancellations, Algorithm 7b can be skipped when we add a column to $\mathbf{A}$ since $\overline{\mathcal{M}}_j = \underline{\mathcal{M}}_j$ for each j. Similarly, when a column is removed from $\mathbf{A}$, Algorithm 7a can be skipped since $\overline{\mathcal{M}}_j = \mathcal{M}_j$ for each j. Hence, when Algorithm 7a followed by Algorithm 7b are applied to a matrix of the form $\mathbf{A}\mathbf{A}^\mathsf{T}$, only Algorithm 7a takes effect during a update while only Algorithm 7b takes effect during a downdate. Thus the approach we have presented in this section for an arbitrary symmetric positive definite matrix generalizes the earlier approach where we focus on matrices of the form $\mathbf{A}\mathbf{A}^\mathsf{T}$.

7 Implementation issues

In this section, we discuss implementation issues in the context of updates and downdates to a matrix of the form $\mathbf{A}\mathbf{A}^\mathsf{T}$. A similar discussion applies to a general symmetric positive definite matrix. We assume that the columns of the matrix $\mathbf{A}$ in the product $\mathbf{A}\mathbf{A}^\mathsf{T}$ are all chosen from among the columns of some fixed matrix $\mathbf{B}$. The update and downdate algorithms can be used to compute the modification to a Cholesky factor corresponding to additions or deletions to the columns of $\mathbf{A}$. The Cholesky factorization of the initial $\mathbf{A}\mathbf{A}^\mathsf{T}$ (before any columns are added or deleted) is often preceded by a fill reducing permutation $\mathbf{P}$ of the rows and columns.

For example, we could compute a permutation to reduce the fill for $\mathbf{B}\mathbf{B}^\mathsf{T}$ since the Cholesky factors of $\mathbf{A}\mathbf{A}^\mathsf{T}$ will be at least as sparse as those of $\mathbf{B}\mathbf{B}^\mathsf{T}$ by Proposition 3.2, regardless of how the columns of $\mathbf{A}$ are chosen from the columns of $\mathbf{B}$. Based on the number of nonzeros in each column of the Cholesky factors of $\mathbf{B}\mathbf{B}^\mathsf{T}$, we could allocate a static storage structure that will always contain the Cholesky factors of each $\mathbf{A}\mathbf{A}^\mathsf{T}$. On the other hand, this could lead to wasted space if the number of nonzeros in the Cholesky factors of $\mathbf{A}\mathbf{A}^\mathsf{T}$ are far less than the number of nonzeros in the Cholesky factors of $\mathbf{B}\mathbf{B}^\mathsf{T}$. Alternatively, we could store the Cholesky factor of the current $\mathbf{A}\mathbf{A}^\mathsf{T}$ in a smaller space, and reallocate storage during the updates and downdates, based on the changes in the nonzero patterns.

The time for the initial Cholesky factorization of $\mathbf{A}\mathbf{A}^\mathsf{T}$ is given by (see [8]) the following expression:

$$O\left(\sum_{j=1}^{m} |\mathcal{L}_j|^2\right),$$

which is $O(m^3)$ if $\mathbf{L}$ is dense. Each update and downdate proceeds by first finding the new symbolic factors and the required path ($\overline{\mathcal{P}}(k)$ for updating, and $\mathcal{P}(k)$ for downdating), using Algorithm 3 or 4 (modified to skip in constant time any column $\mathcal{L}_j^\sharp$ that does not change). Algorithms 5 and 6 are then applied to the columns in the path. The lower triangular solve of the system $\mathbf{L}\mathbf{v} = \mathbf{w}$ in Algorithm 5 can be done in time

$$O\left(\sum_{j\in\mathcal{P}(k)} |\mathcal{L}_j|\right)$$

during downdating, and

$$O\left(\sum_{j\in\overline{\mathcal{P}}(k)} |\mathcal{L}_j|\right)$$

during updating [14]. The remainder of Algorithm 5 takes time $O(|\mathcal{P}(k)|)$ during downdating, and $O(|\overline{\mathcal{P}}(k)|)$ during updating. Our sparse form of Algorithm 6 discussed in §5.3, which computes the product $\overline{\mathbf{L}} = \mathbf{L}\mathbf{G}$, takes time

$$O\left(\sum_{j\in\mathcal{P}(k)} |\mathcal{L}_j|\right)$$

during downdating, and

$$O\left(\sum_{j\in\overline{\mathcal{P}}(k)} |\overline{\mathcal{L}}_j|\right)$$

during updating. Since $\mathcal{L}_j \subseteq \overline{\mathcal{L}}_j$ during an update, it follows that the asymptotic time for the entire downdate or update, both symbolic and numeric, is equal to the

asymptotic time of Algorithm 6. This can be much less than the $O(m^2)$ time taken by Algorithms 5 and 6 in the dense case.

8 Experimental results

We have developed Matlab codes to experiment with all the algorithms presented in this paper, including the algorithms of §6 for a general symmetric, positive definite matrix. In this section, we present the results of a numerical experiment with a large sparse optimization problem from Netlib [3]. The computer used for this experiment was a Model 170 UltraSparc, equipped with 256MB of memory, and with Matlab Version 4.2c.

8.1 Experimental design

We selected an optimization problem from airline scheduling (DFL001). Its constraint matrix $\mathbf{B}$ is 6071-by-12,230 with 35,632 nonzeros. The matrix $\mathbf{BB}^\mathsf{T}$ has 37,923 nonzeros in its strictly lower triangular part. Its Cholesky factor $\mathbf{L}_B$ has 1.49 million nonzeros (with a fill-minimizing permutation $\mathbf{P}_B$ of the rows of $\mathbf{B}$, described below) and requires 1.12 billion floating-point operations and 115 seconds to compute (the LP Dual Active Set Algorithm does not require this matrix, however, as noted earlier, this is an upper bound on the number of nonzeros that can occur during the execution of the LP DASA). This high level of fill-in in $\mathbf{L}_B$ is the result of the highly irregular nonzero pattern of $\mathbf{B}$. The basis matrix $\mathbf{A}_0$ corresponding to an optimal solution of the linear programming problem has 5,446 columns.

We wrote a set of Matlab scripts that implements our complete Cholesky update/downdate algorithm, discussed in §7. We first found $\mathbf{P}_B$, using 101 trials of Matlab's column multiple minimum degree ordering algorithm (`colmmd` [12]), 100 of them with a different random permutation of the rows of $\mathbf{B}$. We then took the best permutation found. With this permutation ($\mathbf{P}_B$), the factor $\mathbf{L}$ of $\mathbf{A}_0\mathbf{A}_0^\mathsf{T}$ has 831 thousand nonzeros, and took 481 million floating-point operations and 51 seconds to compute (using Matlab's `chol`). Following the method used in LP DASA, we added 10^{-12} to the diagonal to ensure positive definiteness. We used the same permutation $\mathbf{P}_B$ for the entire experiment. The initial symbolic factorization took 15 seconds (Algorithm 2). It is this matrix and its factor that are required by the LP DASA.

We did not use Matlab's sparse matrix data structure, since Matlab removes explicit zeros. Changing the nonzero pattern by a single entry can cause Matlab to make a new copy of the entire matrix. This would defeat the asymptotic performance of our algorithms. Instead, the column-oriented data structure we use for $\mathcal{L}$, $\mathcal{L}^\sharp$, and $\mathbf{L}$ consists of three arrays of length $|\mathbf{L}_B|$, an array of length m that contains indices to the first entry in each column, and an array of length m holding the number of

nonzeros in each column. The columns are allocated so that each column can hold as many nonzeros as the corresponding column of $\mathbf{L}_B$, without reallocation.

Starting with the optimal basis of 5,446 columns, we added one column at a time until the basis included all 12,230 columns, and then removed them one at a time to obtain the optimal basis again. The total time and work required to modify the factors was 76,598 seconds and 61.5 billion floating point operations. This divides into

(a) 441 seconds for bookkeeping to keep track of the basis set,
(b) 6,051 seconds for the symbolic updates and downdates (Algorithms 3 and 4),
(c) 21,167 seconds to solve $\mathbf{Lv} = \mathbf{w}$ (in Algorithm 5),
(d) 4,977 seconds for the remainder of Algorithm 5, and
(e) 43,962 seconds for Algorithm 6.

Algorithm 6 clearly dominates the modification algorithm. The symbolic updates and downdates took very little time, even though Algorithms 3 and 4 are not well suited to implementation in Matlab. By comparison, using the Cholesky factorization to solve a linear system $\mathbf{LL}^{\mathsf{T}}\mathbf{x} = \mathbf{b}$ with a dense right-hand-side $\mathbf{b}$ (using our column-oriented data structure for $\mathbf{L}$) at each step took a total of 93,418 seconds and 67.7 billion floating point operations (each solve takes $O(|\mathcal{L}|)$ time). Note that this is much higher than the time taken to solve $\mathbf{Lv} = \mathbf{w}$ in Algorithm 5, because $\mathbf{v}$ and $\mathbf{w}$ are sparse. The time taken for the entire update/downdate computation would be much smaller if our code was written in a compiled language. Solving one system $\mathbf{LL}^{\mathsf{T}}\mathbf{x} = \mathbf{b}$ with a dense right hand side (using the factorization of the optimal basis) takes 5.53 seconds using our column-oriented data structure, 1.26 seconds using a Matlab sparse matrix for $\mathbf{L}$, and 0.215 seconds in Fortran 77.

8.2 Numerical accuracy

In order to measure the error in the computed Cholesky factorization, we evaluated the difference $\|\mathbf{AA}^{\mathsf{T}} - \hat{\mathbf{L}}\hat{\mathbf{L}}^{\mathsf{T}}\|_1$ where $\hat{\mathbf{L}}\hat{\mathbf{L}}^{\mathsf{T}}$ is the computed Cholesky factorization. For the airline scheduling matrix of §8, $\hat{\mathbf{L}}$ has up to 1.49 million nonzeros and it is impractical to compute the product $\hat{\mathbf{L}}\hat{\mathbf{L}}^{\mathsf{T}}$ after each update. To obtain a quick and accurate estimate for $\|\mathbf{E}\|_1$, where $\mathbf{E} = \mathbf{AA}^{\mathsf{T}} - \hat{\mathbf{L}}\hat{\mathbf{L}}^{\mathsf{T}}$, we applied the strategy presented in [16] (see [17, p. 139] for a symbolic statement of the algorithm) to estimate the 1-norm of the inverse of a matrix. That is, we used a gradient accent approach to compute a local maximum for the following problem:

$$\max\{\|\mathbf{Ex}\|_1 : \|\mathbf{x}\|_1 = 1\}.$$

Since $\hat{\mathbf{L}}$ is used multiple times in the following algorithm, we copied our data structure for $\hat{\mathbf{L}}$ into a Matlab sparse matrix.

Algorithm 8 (Estimate 1-norm of an m by m matrix E)

$x_i = 1/m$ for $1 \le i \le m$
$\rho = 0$ (ρ is the current estimate for $\|\mathbf{E}\|$)
while $\|\mathbf{Ex}\|_1 > \rho$ **do**
  $\rho = \|\mathbf{Ex}\|_1$
  for $i = 1$ to m **do**
    $y_i = \quad 1$ if $(\mathbf{Ex})_i \ge 0$
    $y_i = -1$ if $(\mathbf{Ex})_i < 0$
  end for
  $\mathbf{z} = \mathbf{E}^\mathsf{T}\mathbf{y}$
  $j = \arg\max \{|z_i| : i = 1 \text{ to } m\}$
  if $|z_j| \le \mathbf{z}^\mathsf{T}\mathbf{x}$ **return**
  $x_i = 0$ for $i = 1$ to n
  $x_j = 1$
end while
end Algorithm 8

To improve the accuracy of the 1-norm estimate, we used Algorithm 8 three times. In the second and third trials, a different starting vector $\mathbf{x}$ was used as described in [16]. Observe that Algorithm 8 only makes use of the product between the matrix $\mathbf{E}$ and a vector. This feature is important in the context of sparse matrices since $\mathbf{E}$ contains the term $\hat{\mathbf{L}}\hat{\mathbf{L}}^\mathsf{T}$ and it is impractical to compute the product $\hat{\mathbf{L}}\hat{\mathbf{L}}^\mathsf{T}$, but it is practical to multiply $\hat{\mathbf{L}}\hat{\mathbf{L}}^\mathsf{T}$ by a vector. For the airline scheduling matrix of §8 the values for $\|\mathbf{E}\|_1$ initially, at step 6,784, and at the end were 2.49×10^{-13}, 1.01×10^{-10}, and 1.54×10^{-10} respectively. The estimates obtained using Algorithm 8 were identical at the same three steps. On the other hand, the times to compute $\|\mathbf{E}\|_1$ at the initial step and at step 6,784 were 137.9 and 279.5 seconds, while the times for three trials of Algorithm 8 were 8.7 and 14.7 seconds, respectively (excluding the time to construct the Matlab sparse matrix for $\hat{\mathbf{L}}$). Our methods were quite accurate for this problem. After 6,784 updates and 6,784 downdates, or 13,568 changes in $\mathbf{A}$, the 1-norm of $\mathbf{E}$ increased by only a factor 618! Figure 1 shows the estimated value of $\|\mathbf{E}\|_1$ computed every 10 steps using Algorithm 8. The 1-norm of the matrix $\mathbf{AA}^\mathsf{T}$ increases from 458.0 initially to 1107.0 at iteration 6,784, then returns to 458.0 at iteration 13,568. Hence, the product of the computed Cholesky factors agrees with the product $\mathbf{AA}^\mathsf{T}$ to about 15 significant digits initially, while the products agree to about 12 significant digits after 13,568 modifications of $\mathbf{A}$.

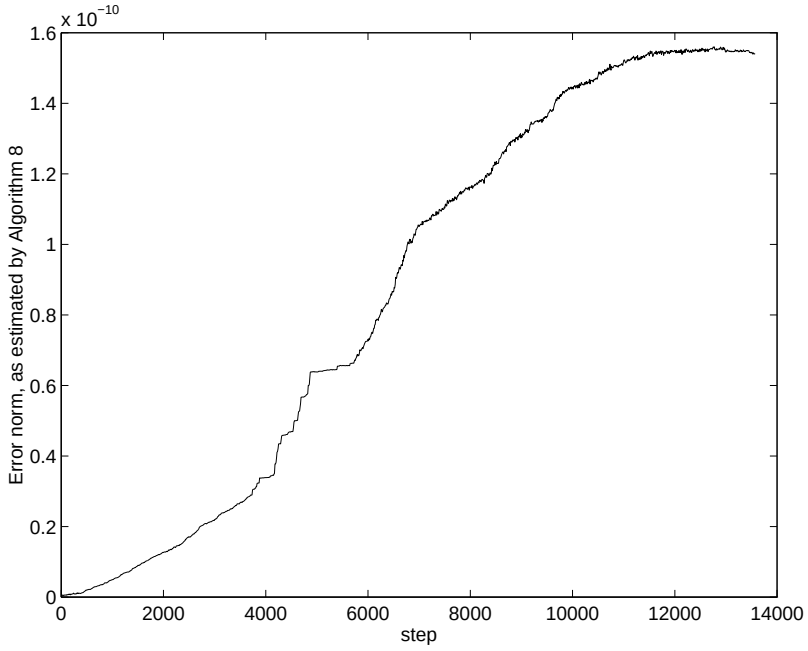

Figure 1: Estimated 1-norm of error in Cholesky factorization

8.3 Alternative permutations

Our methods are optimal in the sense that they take time proportional to the number of nonzero entries in $\mathbf{L}$ that change at each step. However, they are not optimal with respect to fill-in, since we assume a single initial permutation, and no subsequent permutations. A fill-reducing ordering of $\mathbf{B}\mathbf{B}^\mathsf{T}$ might not be the best ordering to use for all basis matrices. A simple pathological example is the m-by-n matrix $\mathbf{B}$, where $n = m(m-1)/2$, and the nonzero pattern of each of the column of $\mathbf{B}$ is a unique pair of integers from the set $\{1, 2, \cdots, m\}$. In this case, every element of $\mathbf{B}\mathbf{B}^\mathsf{T}$ is nonzero, while the nonzero pattern of $\mathbf{A}\mathbf{A}^\mathsf{T}$ is arbitrary. As the matrix $\mathbf{A}$ changes, it might be advantageous to compute a fill-reducing ordering of $\mathbf{A}\mathbf{A}^\mathsf{T}$ if the size of its factors grow "too large." A refactorization with the new permutation would then be required.

We found a fill-reducing permutation $\mathbf{P}_A$ of the optimal basis $\mathbf{A}_0\mathbf{A}_0^\mathsf{T}$ (again, the best of 101 trials of `colmmd`). This results in a factor $\mathbf{L}$ with 381 thousand nonzeros, requiring only 169 million floating point operations to compute. This is significantly less than the number of nonzeros (831 thousand) and floating point operations (481 million) associated with the fill reducing permutation for $\mathbf{B}\mathbf{B}^\mathsf{T}$. We also computed an ordering of $\mathbf{A}\mathbf{A}^\mathsf{T}$ at each step, using `colmmd` just once, and then computed the number of nonzeros in the factor if we were to factorize $\mathbf{A}$ using this permutation ($\mathbf{P}_s$). Although it only takes about 1 second to compute the ordering [12] and symbolic

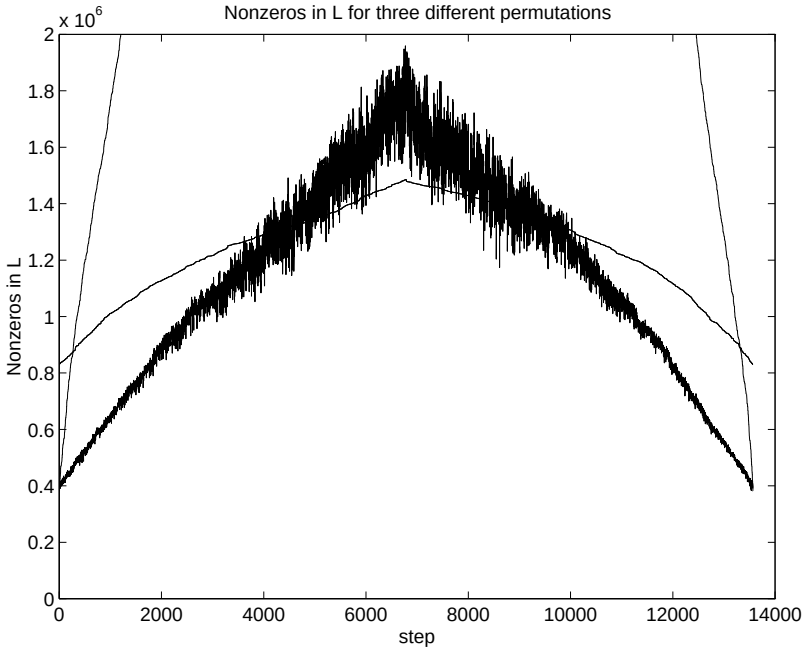

Figure 2: Nonzeros in $\mathbf{L}$ using three different permutations

factorization [13], it is not practical to use 101 random trials at each step.

Figure 2 depicts the nonzero counts of $\mathbf{L}$ for these three different permutations, at each of the 13,568 steps. The fixed permutation $\mathbf{P}_B$ results in the smooth curve starting at 831 thousand and peaking at 1.49 million. The fixed permutation $\mathbf{P}_A$ results in a number of nonzeros in $\mathbf{L}$ that starts at 381 thousand and rises quickly, leaving the figure at step 1,206 and peaking at 7.4 million in the middle. It surpasses $\mathbf{P}_B$ at step 267. Using a permutation, $\mathbf{P}_s$, computed at each step s, gives the erratic line in the figure, starting at 390 thousand and peaking at 1.9 million in the middle. These results indicate that it might be advantageous to start with the fixed permutation $\mathbf{P}_A$, use it for 267 steps, and then refactorize with the permutation $\mathbf{P}_s$ computed at step 267. This results in a new factor with only 463 thousand nonzeros. Near the center of the figure, however, the basis $\mathbf{A}$ includes most of the columns in $\mathbf{B}$, and in this case the $\mathbf{P}_B$ permutation should be used.

9 Summary

We have presented a new method for updating and downdating the factorization $\mathbf{LL}^\mathsf{T}$ of a sparse symmetric positive definite matrix $\mathbf{AA}^\mathsf{T}$. Our experimental results show that the method should be fast and accurate in practice. Extensions to an arbitrary sparse symmetric positive definite matrix, $\mathbf{M}$, have been discussed. We mention here

an additional extension to our work that would be useful.

We do not make use of the supernodal form of the factorization, nor do we use the related compressed pattern of $\mathbf{L}$ [8]. Any method can be used for the numerical factorization of the first basis matrix, of course, but the factor would then be copied into a simple column-oriented data structure. Keeping the supernodal form has the potential of reducing time taken by the symbolic factorization (Algorithm 2), the symbolic update and downdate (Algorithms 3 and 4), and the numerical update and downdate (dense matrix kernels could be used in Algorithms 5 and 6). However, supernodes would merge during update, and split during downdate, complicating the supernodal form of the factorization.

References

[1] P. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Applic., 17 (1996), pp. 886–905.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusets, and McGraw-Hill, New York, 1990.

[3] J. J. Dongarra and E. Grosse, *Distribution of mathematical software via electronic mail*, Comm. ACM, 30 (1987), pp. 403–407.

[4] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, London: Oxford Univ. Press, 1986.

[5] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman, *Yale sparse matrix package, I: The symmetric codes*, Internat. J. Numer. Methods Eng., 18 (1982), pp. 1145–1151.

[6] A. George and J. W. H. Liu, *The design of a user interface for a sparse matrix package*, ACM Trans. Math. Softw., 5 (1979), pp. 139–162.

[7] ——, *An optimal algorithm for symbolic factorization of symmetric matrices*, SIAM Journal on Computing, 9 (1980), pp. 583–593.

[8] ——, *Computer Solution of Large Sparse Positive Definite Systems*, Englewood Cliffs, New Jersey: Prentice-Hall, 1981.

[9] A. George, J. W. H. Liu, and E. Ng, *A data structure for sparse QR and LU factorizations*, SIAM J. Sci. Statist. Comp., 9 (1988), pp. 100–121.

[10] J. R. Gilbert, *Predicting structure in sparse matrix computations*, Tech. Report CS-86-750, Computer Science Dept., Cornell Univ., 1986.

[11] ——, *Predicting structure in sparse matrix computations*, SIAM J. Matrix Anal. Applic., 15 (1994), pp. 62–79.

[12] J. R. Gilbert, C. Moler, and R. Schreiber, *Sparse matrices in MATLAB: design and implementation*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 333–356.

[13] J. R. Gilbert, E. G. Ng, and B. W. Peyton, *An efficient algorithm to compute row and column counts for sparse Cholesky factorization*, SIAM J. Matrix Anal. Applic., 15 (1994), pp. 1075–1091.

[14] J. R. Gilbert and T. Peierls, *Sparse partial pivoting in time proportional to arithmetic operations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 862–874.

[15] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders, *Methods for modifying matrix factorizations*, Mathematics of Computation, 28 (1974), pp. 505–535.

[16] W. W. Hager, *Condition estimates*, SIAM J. Sci. Comput., 5 (1984), pp. 311–316.

[17] ——, *Applied Numerical Linear Algebra*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[18] ——, *Updating the inverse of a matrix*, SIAM Review, 31 (1989), pp. 221–239.

[19] W. W. Hager, C.-L. Shi, and E. O. Lundin, *Active set strategies in the LP dual active set algorithm*, tech. report, Dept. of Mathematics, Univ. of Florida, Gainesville, FL, Aug. 1996. Available at http://www.math.ufl.edu/~hager.

[20] J. W. H. Liu, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Applic., 11 (1990), pp. 134–172.

[21] D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 266–283.

[22] R. Schreiber, *A new implementation of sparse Gaussian elimination*, ACM Trans. Math. Softw., 8 (1982), pp. 256–276.

[23] A. H. Sherman, *On the efficient solution of sparse systems of linear and nonlinear equations*, Tech. Report 46, Yale Univ. Dept. of Computer Science, Dec. 1975.

[24] G. Strang, *Linear Algebra and Its Applications*, Academic Press, New York, 1980.