Main observation. Consider the \( LDL^T \) decomposition of a SPD matrix
\[
A = \begin{bmatrix} B & V^T \\ V & C \end{bmatrix} = \begin{bmatrix} L & 0 \\ V L^{-T} & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & C - V B^{-1} V^T \end{bmatrix} \begin{bmatrix} L^T & L^{-1} V^T \\ 0 & I \end{bmatrix},
\]
where \( B = LL^T \) is the Cholesky factorization of \( B \). The Schur complement update \(-V B^{-1} V^T\) can be written as
\[
-V B^{-1} V^T = -(V L^{-T})(L^{-1} V^T) = -\sum_{k=1}^{j-1} \begin{bmatrix} l_{j,k} \\ l_{j+1,k} \\ \vdots \\ l_{n,k} \end{bmatrix} \begin{bmatrix} l_{j,k} & l_{j+1,k} & \cdots & l_{n,k} \end{bmatrix}.
\]
Here \( j : n \) represents the index of \( VL^{-T} \) in the global matrix. Every term in the summation can be seen as the contribution from the \( k \)-th column.

Frontals  Let \( T[j] \) denotes the descendants of node \( j \) and itself, i.e. nonzero entries in the lower off-diagonal in the symbolic \( L \) component of \( A = LL^T \) and itself. Then we have
\[
\bar{U}_j = -\sum_{k \in T[j] \setminus \{j\}} \begin{bmatrix} l_{j,k} \\ l_{i_1,k} \\ \vdots \\ l_{i_r,k} \end{bmatrix} \begin{bmatrix} l_{j,k} & l_{i_1,k} & \cdots & l_{i_r,k} \end{bmatrix},
\]
where \( j, i_1, i_2, \ldots, i_r \) represents all the possible nonzero patterns for node \( j \) and above.
For example, in Figure, if $j = 6$, we have $T[j] = \{2, 3, 4, 5, 6\}$, $i_1 = 8$, $i_2 = 9$.

The Schur complement, which we call **frontal matrix**, is

$$F_j = \begin{bmatrix}
a_{j,j} & a_{j,i_1} & \cdots & a_{j,i_r} \\
a_{i_1,j} & \ddots & & \\
\vdots & & \ddots & \\
a_{i_r,j} & & & \ddots
\end{bmatrix} + \bar{U}_j$$

It is easy to see

$$\bar{U}_j = - \sum_{k<j, l_k \neq 0} \begin{bmatrix} l_{j,k} \\
l_{i_1,k} \\
\vdots \\
l_{i_r,k}
\end{bmatrix} \begin{bmatrix} l_{j,k} & l_{i_1,k} & \cdots & l_{i_r,k}
\end{bmatrix} - \sum_{k \in T[j] - \{j\}, l_k = 0} \begin{bmatrix} 0 \\
l_{i_1,k} \\
\vdots \\
l_{i_r,k}
\end{bmatrix} \begin{bmatrix} 0 & l_{i_1,k} & \cdots & l_{i_r,k}
\end{bmatrix}$$

It is easy to see that only the first term contributes to the first row/column and therefore,

$$F_j = \begin{bmatrix} l_{j,j} & 0 \\
l_{i_1,j} & 1 & 0 \\
\vdots & & \ddots & \\
l_{i_r,j} & & & I
\end{bmatrix} \begin{bmatrix} 1 & 0 \\
0 & U_j
\end{bmatrix} \begin{bmatrix} l_{j,j} & l_{i_1,j} & \cdots & l_{i_r,j} \\
0 & 0 & \ddots & \\
\vdots & & & I
\end{bmatrix},$$

where

$$U_j = - \sum_{k \in T[j]} \begin{bmatrix} l_{i_1,k} \\
\vdots \\
l_{i_r,k}
\end{bmatrix} \begin{bmatrix} l_{i_1,k} & \cdots & l_{i_r,k}
\end{bmatrix}$$
Frontal Method  We use $\natural$ to define an addition by indexes. If

$$R = \begin{bmatrix} p & q & u & v \end{bmatrix}, S = \begin{bmatrix} w & x \\ y & z \end{bmatrix}$$

where $R$ corresponds to \{5, 8\} and $S$ corresponds to \{5, 9\} then we have

$$R\natural S = \begin{bmatrix} p + w & q & x \\ u & v & 0 \\ y & 0 & z \end{bmatrix}.$$

We can see if $c_1, c_2, \ldots, c_s$ are the children of node $j$ in the elimination tree then

$$F_j = \begin{bmatrix} a_{jj} & a_{ji_1} & \cdots & a_{ji_r} \\ a_{i_1,j} & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ a_{i_r,j} & & & a_{i_r,j} \end{bmatrix} ~ \natural U_{c_1} \natural U_{c_2} \cdots \natural U_{c_s}$$

We have the following algorithm for sparse Cholesky factorization
Algorithm 1 Sparse Cholesky Factorization

1: procedure sparseChol
2:   for $j = 1, 2, \ldots, n$ do
3:     Let $j, i_1, \ldots, i_r$ be the locations of nonzeros in $L_{\star j}$.
4:     Let $c_1, \ldots, c_s$ be the children of $j$ in the elimination tree.
5:     Form the matrix $\tilde{U} = U_{c_1} \circ U_{c_2} \circ \cdots \circ U_{c_s}$ and frontal matrix
6:     $F_j = \begin{bmatrix} a_{jj} & a_{ji_1} & \cdots & a_{ji_r} \\ a_{i_1,j} & & & \\ \vdots & & & 0 \\ a_{i_r,j} \end{bmatrix} \circ \tilde{U}$
7:     Factorize $F_j$ into
8:     $\begin{bmatrix} l_{j,j} & 0 \\ l_{i_1,j} & \vdots \\ \vdots & I \\ l_{i_r,j} & \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & U_j \end{bmatrix} \begin{bmatrix} l_{j,j} & l_{i_1,j} & \cdots & l_{i_r,j} \\ \end{bmatrix}$
9: function findL($A$)
10:   $L \leftarrow \text{tril}(A) \neq 0$
11:   for $i = 1, 2, \ldots, n$ do
12:     for $j \in \text{nonzero}(L_{:,i}) \setminus \{i\}$ do
13:       for $k \in \text{nonzero}(L_{:,i}) \setminus \{i\}, k > j$ do
14:         $L_{kj} \leftarrow \text{true}$
15:   return $L$

It is without any difficulty that the method can be generalized to any matrix
Algorithm 2 Sparse Cholesky Factorization

1: procedure SPARSECHOL
2: for j = 1, 2, ..., n do
3:   Let j, i_1, ..., i_r be the locations of nonzeros in L_{*j}.
4:   Let c_1, ..., c_s be the children of j in the elimination tree.
5:   Form the matrix \( \hat{U} = U_{c_1}\hat{U}U_{c_2}\hat{U} \cdots \hat{U}U_{c_s} \) and frontal matrix
6:   \( f_j = \begin{bmatrix} a_{jj} & a_{ji_1} & \cdots & a_{ji_r} \\ a_{i_1,j} & \vdots & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ a_{i_r,j} \end{bmatrix} \).
7:   Factorize \( F_j \) into

\[
\begin{bmatrix}
1 & 0 \\
l_{i_1,j} & I \\
l_{i_r,j} & 0
\end{bmatrix}
\begin{bmatrix}
u_{j,j} & u_{i_1,j} & \cdots & u_{i_r,j} \\
0 & U_j
\end{bmatrix}
\]

Left-looking, right-looking and multifrontal Recall in LU factorization, after scaling one column, we may apply the addition to the other columns immediately or delay it until we need to use the column. This is called left-looking and right-looking respectively. Multifrontal method is another style of looking, which can be illustrated below.

![Figure 2: Left-looking, right-looking and multifrontal.](image)

General Assembly Tree. We do not have to use elimination tree for the factorization process. Actually, we only need an assembly tree, i.e. for any node \( j \) with parent node \( p \),
the off-diagonal structure of $L_{\star j}$ is a subset of the structure of $L_{\star p}$ and $p > j$. For example, by defining the parent $p$ of $j$

$$p = \min\{i > j \mid \text{off-diagonal structure of } L_{\star j} \subset \text{structure of } L_{\star i}\},$$

we will obtain a valid assembly tree (Figure 3).

![Assembly tree](image)

**Figure 3:** Assembly tree.

**Pivoting and scaling** It is important to do pivoting and scaling for numerical accuracy reasons. As indicated in [1], the relative forward error is bounded by the condition number of the linear system, multiplied by the backward-error.

- an ill-posed problem: the backward error can be small even if the solution is far from the exact solution; in that case, the condition number of the system is large;
- an unstable algorithm, leading to a large backward error compared to the machine precision, even when the condition number of the linear system is small.

They can be measured by

- Backward error

\[
\text{err} = \min\{\epsilon > 0 \text{ such that } \|\Delta A\| \leq \epsilon \|A\|, \|\Delta b\| \leq \epsilon \|b\|\},
\]

\[
(A + \Delta A)\hat{x} = b + \Delta b
\]

\[
= \frac{\|A\|\|\hat{x} - b\|}{\|A\|\|\hat{x}\| + \|b\|}
\]
• condition number

\[ \frac{\|A^{-1}\|A\|\hat{x}\| + \|A^{-1}\|\|b\|}{\|\hat{x}\|} \]

The mathematical formula for pivoting and scaling is (note scaling matrix and permutation matrix are communicable)

\[ PD_r A D_c Q D_c^{-1} x = PD_r b \]

• Doing scaling on a row \( i \) is equivalent to doing the corresponding scaling on all the \( l_{ij} \) components for \( j \in T[i] - \{i\} \), and scaling on \( a_{ij} \) on all the \( j \in \text{Adj}_G(i) \).

• Doing scaling on a column \( i \) is equivalent to doing the corresponding scaling on all the \( u_{ji} \) components for \( j \in T[i] - \{i\} \), and scaling on \( a_{ji} \) on all the \( j \in \text{Adj}_G(i) \).

As for pivoting, we have limited choice as pivoting may deconstruct the elimination trees. The rule here is to only pivot inside the blocks that are already summed. This requires us to add the contribution of child blocks to parents immediately after factorization instead of the left-looking way.

Figure 4: Pivoting on a tree. Node 7 and node 8 have already been summed as the contributions from 1 and 6 have been added. Now if we want to factorize 7, we can do the pivot \( 8 \leftrightarrow 7 \). This is the same as interchange node 7 and 8.

**Supernodal**  To take advantage of higher BLAS level operations, the use of supernodes is preferred. Supernode is defined as\(^2\)
A supernode is a maximal set of contiguous nodes such that

$$\text{Adj}_G(T[j]) = \{j, j+1, \ldots, j+t\} \cup \text{Adj}_G(T[j+t])$$

It follows from this definition that in a supernode $\{j, j+1, \ldots, j+t\}$, for $1 \leq k \leq t$ the node $j+k$ is the parent of $j+k-1$ in the elimination tree.

Here $\text{Adj}_G(T[j])$ gives the subscript set of nonzeros in column $L_{*j}$. A supernode corresponds to a maximal block of contiguous columns in the Cholesky factor, where the corresponding diagonal block is full triangular, and these columns all have identical off-block-diagonal column structure.

**Solve Procedure**  Given a right hand side, we have the following solving algorithm:

---

**Algorithm 3** Forward Elimination

1. `function FORWARD_ELIMINATION(b)`
2. `for i = 1, 2, \ldots, n do`  // Working on front $i$
3. `Solve l_{ii}y_i = b_i`
4. `for k > i, l_{ki} \neq 0 do`
5. `b_k \leftarrow b_k - l_{ki}y_i`

---

**Algorithm 4** Backward Substitution. It is important to use right-looking method to do backward substitution. In every step, we first add the contributions from the successors, i.e. $\text{Adj}_G(\cdot)$.

1. `function BACKWARD_SUBSTITUION(y)`
2. `for i = n, n-1, \ldots, 1 do`  // For SPD, $u_{ik} = l_{ki}$
3. `for k > i, u_{ik} \neq 0 do`
4. `y_i \leftarrow y_i - u_{ik}x_k`
5. `Solve u_{ii}x_i = y_i`

---

**Programming considerations**  On the programming aspect, the data should not live in the graph itself. Instead, it should be stored in a separate data structure. For example, in the example above, the data can be stored in a map which takes (self-)edges as keys.

The following result illustrates the application of multifrontal method to random matrices without pivoting.
As the matrix is random, the fill-in will be terrible and therefore the complexity will be like $O(N^2)$ for the solving step, which is present in the last figure.

References
