Numerical Linear Algebra for Computational Science and Information Engineering

Lecture 06

Introduction to Direct Methods for Sparse Linear Systems

Nicolas Venkovic nicolas.venkovic@tum.de

Group of Computational Mathematics School of Computation, Information and Technology Technical University of Munich

Summer 2025



### Outline

1	Solving sparse triangular linear systems Section 9.3 in Darve & Wotters (2021)	1
2	Cholesky factorization Section 9.4 in Darve & Wotters (2021)	5
3	Nested dissesction Section 9.5 in Darve & Wotters (2021)	20
4	Homework problems	26

# Solving sparse triangular linear systems Section 9.3 in Darve & Wotters (2021)

# When L is sparse and b is dense

- We want to solve Lx = b where L is a sparse lower-triangular matrix with non-zero diagonal entries.
- Remember how to proceed when *L* is dense:

1. 
$$x_1 = b_1/l_{11}$$
  
2.  $x_2 = (b_2 - l_{21}x_1)/l_{22}$   
 $\vdots$   
*i.*  $x_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij}x_j\right)/l_{ii}$ 

*n.* 
$$x_n = \left(b_n - \sum_{j=1}^{n-1} l_{nj} x_j\right) / l_{nn}$$

- When L is sparse, we simply need to skip the zero components l<sub>ij</sub> in each summand.
- ▶ We will see in practice session that this can easily be implemented.
- ► The final form of the implementation depends on the sparse matrix data structure used to store *L*.

# When both L and b are sparse

- When L and b are sparse, then the solution x may be sparse.
- Ideally, we would like to solve for x as follows:

1. for 
$$i = 1, ..., n$$
:

- 2. if  $x_i \neq 0$ :
- 3.  $x_i \leftarrow b_i / \ell_{ii}$
- 4. for j = 1, ..., i 1:
- 5. if  $\ell_{ij} \neq 0$ :

5. 
$$x_i \leftarrow x_i - \ell_{ij} x_j / \ell_{ii}$$

- But iterating over the non-zero components of x requires to know the structure of x.
- For any non-zero  $x_i$ , we have either or both (a)  $b_i \neq 0$ 
  - (b) there is some j < i such that  $\ell_{ij} \neq 0$  and  $x_j \neq 0$ .

#### When both L and b are sparse, cont'd<sub>1</sub>

Let G = (V, E) be the graph associated with L, then we denote by X ⊂ V the minimal set of vertices so that either or both (a) and (b) hold. That is, X ⊂ V is the minimal set such that:

 $b_i \neq 0 \implies i \in X \text{ and } \ell_{ij} \neq 0 \text{ and } j \in X \implies i \in X.$ 

#### Definition (Reachability & Reach)

- A vertex  $i \in V$  in a directed graph G = (V, E) is **reachable from a vertex**  $j \in V$ , if there is a directed path from j to i in G. That is, if there is a sequence of edges  $(j, i_1), (i_1, i_2), \ldots, (i_{k-1}, i_k), (i_k, i)$  where all the edges are in E.
- The set of vertices  $i \in V$  reachable from a vertex  $j \in V$  is the **reach** of j.



# When both L and b are sparse, cont'd<sub>2</sub>

- ► Then, if we let B ⊂ V be the set of vertices i ∈ V such that b<sub>i</sub> ≠ 0, then X is the set of vertices reachable from B.
- Consequently, the set X can be found by operating on the graph G = (V, E) associated with L.

Namely, the set X can be found using a **depth-first traversal** (DFS, i.e., for depth-first search) from every vertex in B.

- Depth-first traversal starts from some node j, and explores as far as possible along each branch in the graph before backtracking.
- We will see an implementation of depth-first traversal in the practice session.
- Procedure to solve Lx = b where both L and b are sparse is as follows:

#### Linear solve of Lx = b where both L and b are sparse

- 1. Define the set B from the sparsity pattern of b.
- 2. Find the set X of non-zero x components using DFS on B.
- 3. Run modified version of the algorithm to solve Lx = b with a sparse L, but compute  $x_i$  only if  $i \in X$ .

# Cholesky factorization Section 9.4 in Darve & Wotters (2021)

# Up-looking Cholesky algorithm

- Now that we know how to solve sparse trangular systems, we can use this to obtain a sparse Cholesky factorization.
- In particular, the up-looking Cholesky algorithm performs a Cholesky factorization by doing a series of sparse triangular solves.
- Proceeding by construction, assume the (n 1)-dimensional leading block L' of the Cholesky factor L of A is already known, leading the following structure of the LL<sup>T</sup> = A factorization:



# Up-looking Cholesky algorithm, cont'd<sub>1</sub>

This leads to the following algorithm:

#### Up-looking Cholesky algorithm

Given a sparse SPD matrix  $A \in \mathbb{R}^{n \times n}$ , initialize  $L' := \sqrt{a_{11}}$ . For  $k = 2, \ldots, n$ :

- Let the leading k-by-k block of A be written as  $\begin{bmatrix} A' & b \\ b^T & a \end{bmatrix}$  where A' is the (k-1)-dimensional leading block, b is (k-1)-by-1 and a is a scalar.
- Solve for  $x \in \mathbb{R}^{k-1}$  such that L'x = b where L' and b are sparse.

- Compute  $w := \sqrt{a - x^T x}$ , and update

$$L' := \begin{bmatrix} L' & 0\\ x^T & w \end{bmatrix}$$

Return L := L'

Consequently, the sparse Cholesky factor L of the sparse matrix A is formed by performing n-1 sparse triangular solves of sizes  $1, \ldots, n-1$ .

nicolas.venkovic@tum.de

NLA for CS and IE – Lecture 06

# Up-looking Cholesky algorithm, cont'd<sub>2</sub>

Consider a matrix A with the following non-zero pattern:



Since the 2-by-2 leading block A' of A is diagonal, so is the corresponding 2-by-2 Cholesky factor L' such that L'L'<sup>T</sup> = A'.

Let the vector  $\begin{bmatrix} A' & b \\ b^T & a \end{bmatrix}$  of A. Then, the up-looking Cholesky algorithm requires that we do the sparse triangular solve of L'x = b. The reach of node 1



NLA for CS and IE - Lecture 06

# Up-looking Cholesky algorithm, cont'd<sub>3</sub>

• Consider a matrix A with the following non-zero pattern:



• Let the vector 
$$\Box_{b}$$
 complete the 4-by-4 leading block  $\begin{bmatrix} A' & b \\ b^T & a \end{bmatrix}$  of  $A$ .  
Then, the up-looking Cholesky algorithm requires that we do the sparse triangular solve of  $L'x = b$ .  
The reach of node 1 in the graph associated with  $L'$  is 1 and 3

The Cholesky factor  $L' := \begin{bmatrix} L' & 0 \\ x^T & w \end{bmatrix}$  of  $\begin{bmatrix} A' & b \\ b^T & a \end{bmatrix}$  has structure

NLA for CS and IE – Lecture 06

# Up-looking Cholesky algorithm, cont'd<sub>4</sub>

Consider a matrix A with the following non-zero pattern:





Then, the up-looking Cholesky algorithm requires that we do the sparse triangular solve of L'x = b. The reach of nodes



NLA for CS and IE - Lecture 06

# Up-looking Cholesky algorithm, cont'd\_5

The up-looking Cholesky algorithm yield a factor with the following non-zero pattern:



Note that the sparsity of L resembles that of A, with additional fill-ins:



- While the up-looking algorithm is better than performing a dense Cholesky factorization, it does require many DFS in graphs.
- ► We'll now try to do better than the up-looking algorithm.

NLA for CS and IE - Lecture 06

### Elimination tree

- The graph associated with the sparsity pattern of a Cholesky factor L has a special property which allows to ignore many of its edges and retain the same reach.
- Consider what happens when we ignore all the non-zero entries of L below the first subdiagonal non-zero component. E.g.,



Removing these entries results in a **sparsification of the associated graph**:



#### Elimination tree, cont'd



Two general properties are observed:

- In the reach of every vertex remains unchanged by sparsification.
- ② Every vertex of the sparsified graph has at most one edge leading out of it. I.e., if the graph is connected, then it is a directed tree.

Remarks:

- The sparsified graph is called an elimination tree.
- The elimination tree **may be disconnected**, in which case it is a forest, but even then, it will be called an elimination tree.
- The elimination tree is an important data structure that **can be used to simplify** all **reach calculations** in a sparse Cholesky factorization.

### Non-zero pattern of $\boldsymbol{L}$

- Say we aim to compute the *i*-th line of the Cholesky factor L of an SPD A.
- We are equipped with L' := L[1:i-1,1:i-1]:



- ▶ We saw the non-zero entries of L[i, 1 : i 1] are the non-zero entries of the solution x of the above system with right-hand side b := A[1 : i - 1, i].
- Remember from our sparse triangular solves, x<sub>j</sub> = ℓ<sub>ij</sub> is non-zero either if
   (a) b<sub>j</sub> = a<sub>ij</sub> ≠ 0, or if (b) ∃k < j so that both ℓ<sub>jk</sub> ≠ 0 and x<sub>k</sub> = ℓ<sub>ik</sub> ≠ 0.

# Non-zero pattern of L, cont'd

▶ Therefore, the pattern of non-zero values of *L* is characterized as follows:

Graph of (possible) non-zero entries of LLet j < i, then  $\ell_{ij}$  is non-zero if (a)  $a_{ij} \neq 0$ , or (b) there is a column index k < j such that  $\ell_{jk}$  and  $\ell_{ik}$  are non-zero. We denote by  $G_{ch}$  the graph with fewest edges that respect (a) and (b). That is,  $G_{ch}$  is the minimal graph such that  $a_{ij} \neq 0 \implies (j,i) \in G_{ch}$  and  $(j,k), (i,k) \in G_{ch} \implies (j,i) \in G_{ch}$ .

The graph G<sub>ch</sub> is a superset of the non-zero pattern of the Cholesky factor L of A.

It can be that  $(j,i) \in G_{ch}$  but  $\ell_{ij}$  numerically cancels out. However, if so, a tiny perturbation of A with fixed sparsity is enough to make  $\ell_{ij} \neq 0$ . Therefore, the graph  $G_{ch}$  is best referred to as the graph of possible non-zero entries of the Cholesky factor L.

# Definition of the elimination tree

The elimination tree can be defined as follows:

#### Elimination tree

Let A be an SPD matrix, and  $G_{ch}$  be the graph representing the non-zero entries of the Cholesky factor L of A.

The elimination tree is obtained as follows.

For each node i in  $G_{ch}$ :

- Let  $V_i$  be the set of nodes j of  $G_{ch}$  for which there is an edge  $(i, j) \in G_{ch}$ , i.e.,  $V_i$  is the set of out-neighbors of i. Let  $p_i = \min V_i$  be the smallest-indexed node in  $V_i$ .
- Remove the edges (i, j) for all  $j \in V_i \setminus \{p_i\}$  from  $G_{ch}$ , i.e., remove all the the edges leaving i except for  $(i, p_i)$ .

The elimination tree is what's left of  $G_{ch}$ .

Example:

#### Properties of the elimination tree

- Since for each vertex i in G<sub>ch</sub>, the elimination tree is formed by removing all but one out-neighbors, each vertex is left with at most one single out-neighbor, and the elimination tree is indeed a tree, or at least a forest.
- ► Consider the following example for a graph G<sub>ch</sub> of non-zero entries of the Cholesky factor L:



As  $G_{ch}$  is, the reach of k is j, i.

If k < j < i, the elimination tree is formed by removing the edge (k, i). Then, the reach of k in the elimination tree is still j, i.

#### Theorem (Conservation of reach)

For a given graph  $G_{ch}$  of non-zero entries of a Cholesky factor L of A, for any  $1 \le i \le n$ , the reach of the corresponding elimination tree is the same as the reach of i in  $G_{ch}$ .

#### Computing the elimination tree from A

Since the elimination tree has the same reach as G<sub>ch</sub>, but is sparser than G<sub>ch</sub>, it can be used to more efficiently identify the non-zero entries of the Cholesky factor.

For that, we need to figure out how to efficiently compute the elimination tree from the given sparsity pattern of A.

The idea behind computing the elimination tree of A is to proceed one vertex at a time, maintaining a forest which contains all the vertices added so far. The elimination tree shall be obtained once all the vertices are added.

Suppose we have a forest which has all the vertices 1,..., i − 1 at the correct place. To proceed with the *i*-th vertex, if a<sub>ik</sub> ≠ 0 for some k < i, then we'll want i to be in the reach of k. In order to avoid potential redundant edges, we should then connect i to whichever vertex j which is at the leaf of the tree containing k.</p>

# Computing the elimination tree from A, cont'd

- The pseudocode of the algorithm to build the elimination tree from the sparsity pattern of A is given by
  - 1. Initialize a forest  $\mathcal{F}=\emptyset$  :
  - 2. For i = 1, ..., n:
  - 3. Add vertex i to  $\mathcal{F}$
  - 4. For all k < i such that  $a_{ik \neq 0}$ :
  - 5. Find vertex j at the leaf of k's tree
  - 6. Add the edge (j, i) to  $\mathcal{F}$

Taking the same sparse matrix A as earlier, the elimination tree is then built as follows:



We see that the same elimination tree is obtained as before.

### Summary

To compute a sparse Cholesky factor  $\boldsymbol{L}$  of a sparse matrix  $\boldsymbol{A},$  we

- Build the elimination tree of A, at cost  $\mathcal{O}(|A|)$ , where |A| is the number of non-zero entries in A.
- **②** Find the graph  $G_{ch}$  of possible non-zero entries of L using reaches of the elimination tree.
- $\bigcirc$  Perform the up-looking Cholesky factorization to build L.

Pseudocode of the up-looking Cholesky factorization to build row k of L:

- 1. L[k, 1:k] := A[k, 1:k]
- 2. For each j < k such that  $\ell_{kj} \neq 0$ :
- 3.  $\ell_{kj} \leftarrow \ell_{kj}/\ell_{jj}$
- 4. For each i > j such that  $\ell_{ij} \neq 0$ :

5. 
$$\ell_{ki} \leftarrow \ell_{ki} - \ell_{ij}/\ell_{kj}$$

# Nested dissesction Section 9.5 in Darve & Wotters (2021)

# Reducing fill-ins in ${\cal L}$

While the row and column permutations of a matrix do not really impact the solution of a linear system (i.e.,  $P_rAP_c \cdot P_c^T x = P_r b$ ), they can have a significant impact on the sparsity pattern, i.e., the graph  $G_{ch}$  of the Cholesky factor:



Even though only one row and one column are permuted between  $A_1$  and  $A_2$ , the difference between the numbers of fill-ins in  $L_1$  and  $L_2$  is very significant.

How should a matrix be ordered to reduce the number of fill-ins in the Cholesky factor L?

nicolas.venkovic@tum.de

NLA for CS and IE - Lecture 06

#### One step of nested dissection

- Nested dissection is a strategy for ordering a matrix A in a way that closely minimizes the number of fill-ins in L.
- Nested dissection is a recursive method based on graph partitioning.
- Consider the symmetric matrix A with an associated graph G.

Let the vertices of G be decomposed in the disjoint union of  $V_1, V_2$  and S, so that there are no edges between vertices of  $V_1$  and  $V_2$ .

If  ${\cal G}_1$  and  ${\cal G}_2$  are the induced graph on  $V_1$  and  $V_2,$  respectively, then we have



where S is referred to as a **separator**.

21/26

### One step of nested dissection, $cont'd_1$

- ▶ A node separator set *S* partitions the graph *G* of *A* into three disjoint sets of vertices *V*<sub>1</sub>, *V*<sub>2</sub> and *S* such that none of the nodes of *V*<sub>1</sub> are connected to any of the nodes of *V*<sub>2</sub>, and vice-versa.
- ▶ The removal of *S* from the graph *G* leads to two subgraphs *G*<sub>1</sub> and *G*<sub>2</sub>, disconnected from each other.
- Consider what happens when we order the vertices as

(vertices of  $G_1$ , vertices of  $G_2, S$ )

we obtain the matrix



which is structurally close to the  $A_1$  matrix with small number of fill-ins.

Α

nicolas.venkovic@tum.de

NLA for CS and IE - Lecture 06

#### One step of nested dissection, cont'd<sub>2</sub>



As a result of the block diagonal structure due to  $G_1$  and  $G_2$ , the Cholesky factor L of the reordered matrix will preserve a block diagonal structure. If the blue blocks are dense, the sparsity of L is exactly given by that of A. In general, each block of L will be sparse.

From here, we will proceed as follows:

- Entries in S. For these entries, we give up and accept whatever fill-ins happen. Thus we want S to be as small as possible.
- ▶ Entries in G<sub>1</sub> and G<sub>2</sub>. For these entries, we will recurse on the blocks G<sub>1</sub> × G<sub>1</sub> and G<sub>2</sub> × G<sub>2</sub>, i.e., find small separators S<sub>1</sub> and S<sub>2</sub> for G<sub>1</sub> and G<sub>2</sub>, respectively, and so on.

# Nested dissection

The basic idea of nested dissection is to recursively apply the procedure we just described, and yield a nested dissection ordering of the graph

```
nodes.
            Pseudocode for the nested dissection algorithm
            function nested_dissection(G::Graph)
                # returns a nested dissection ordering of the nodes of G
                if size(G) < threshold
                    # This is the base case of the recursion
                    return nodes(G) # nodes of graph G
                end
                [G1, G2, S] = find_separator_set(G)
                # G1 and G2 are two disconnected subgraphs
                # S is the node separator set
                P1 = nested_dissection(G1)
                P2 = nested dissection(G2)
                return [P1: P2: S]
            end
```

The description of find\_separator\_set is beyond the scope of this class. A good separator set has as few nodes as possible, and it decomposes the graph in roughly equally sized subgraphs. Finding a good separator set is actually a NP-hard problem.

nicolas.venkovic@tum.de

NLA for CS and IE - Lecture 06

### Nested dissection, cont'd

In the matrix, the recursive process of nested dissection looks like this:



As we can see, if good separators are chosen, the "down-and-right-arrow" patterns shows up at all scales, and we can guarantee that more and more entries of L will be zero.

# Homework problems

#### Homework problem

Turn in **your own** solution to the following problem:

**Pb.16** Find the non-zero pattern of the Cholesky factor *L* for the following matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & a_{14} \\ a_{12} & a_{22} & 0 & a_{24} \\ 0 & 0 & a_{33} & 0 \\ a_{14} & a_{24} & 0 & a_{44} \end{bmatrix}$$

Show your work using the up-looking Cholesky factorization algorithm.

٠