CEE6501 — Lecture 4.3

Matrix Sparsity and Bandwidth

Learning Objectives

By the end of this lecture, you will be able to:

- Explain why global stiffness matrices for trusses are typically **sparse**
- Define **half-bandwidth** and relate it to storage and computational cost
- Show how **node numbering** changes bandwidth without changing physics

Agenda

1. Sparsity and bandwidth in the global stiffness matrix

Part 1 — Sparsity and Bandwidth

Sparsity (local physical connectivity)

In truss and frame models, each node connects to only a small number of neighboring elements.

As a result, each equilibrium equation involves only a few degrees of freedom. Entries in the stiffness matrix only exist where DOFs are linked by members.

This locality leads to a **sparse** global stiffness matrix $\mathbf{K}$: most entries are exactly zero.

- Sparsity enables memory-efficient storage
- It also enables specialized **sparse solvers** that avoid unnecessary operations

Sparsity is a direct consequence of the *physics and topology* of the structure.

Bandwidth (connectivity meets indexing)

Sparsity describes *how many* nonzero entries exist. **Bandwidth** describes *how far from the diagonal* those nonzeros extend.

For a symmetric matrix $\mathbf{K}$, the **half-bandwidth** is defined as

$$NHB = \max\{|i-j| : K_{ij} \neq 0\}.$$

A smaller half-bandwidth means that nonzero entries are clustered closer to the diagonal.

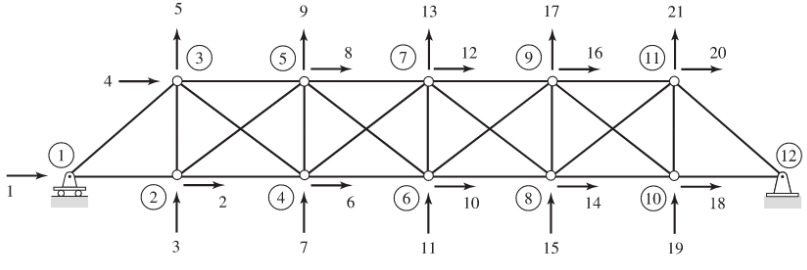
Why node numbering matters

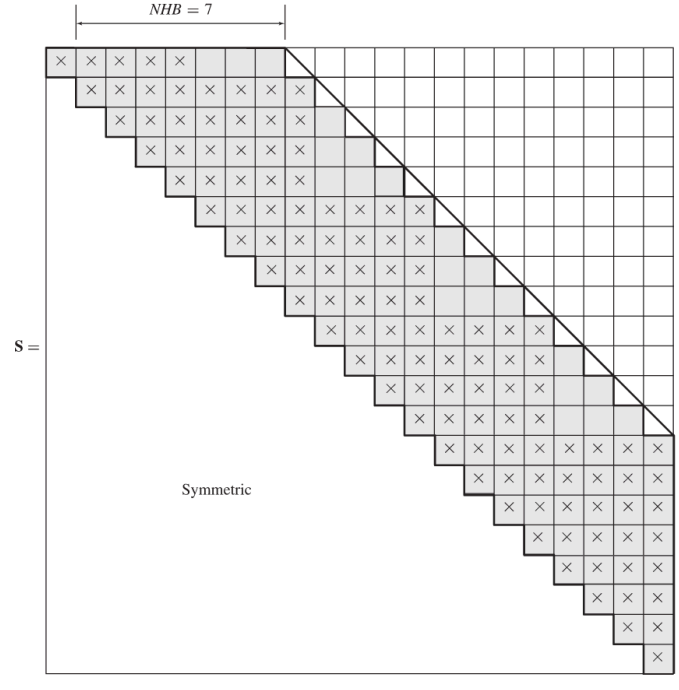
The physical structure is unchanged, but the **algebraic representation** of $\mathbf{K}$ depends on the ordering of degrees of freedom.

- Poor numbering places strongly coupled DOFs far apart → **large bandwidth**
- Good numbering keeps coupled DOFs close → **small bandwidth**

Bandwidth affects the cost of matrix factorization, even though it has no effect on the underlying physics.

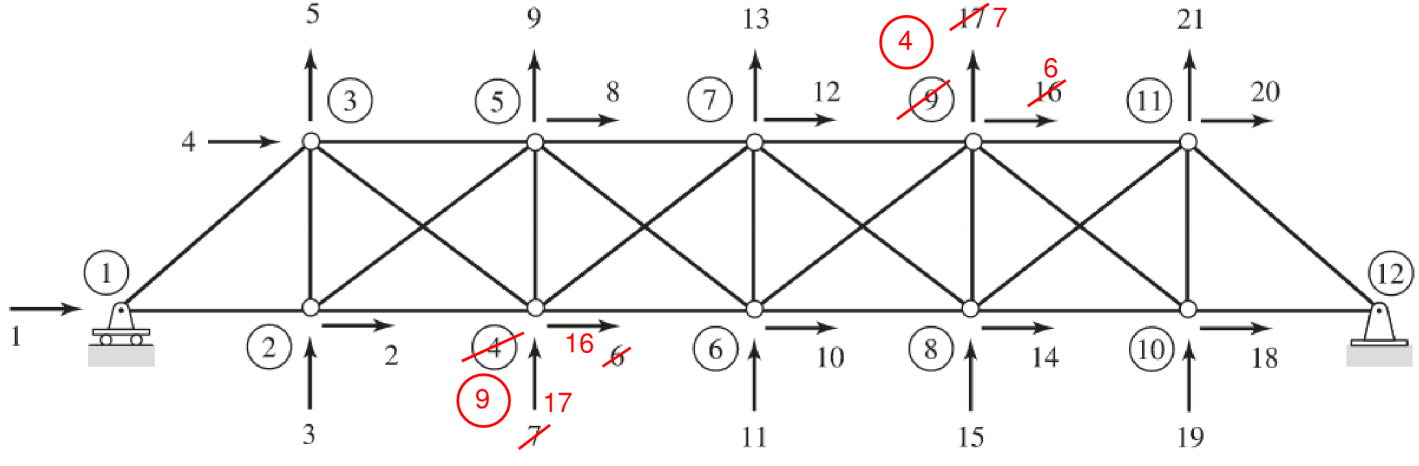
Example 1: A well-numbered system



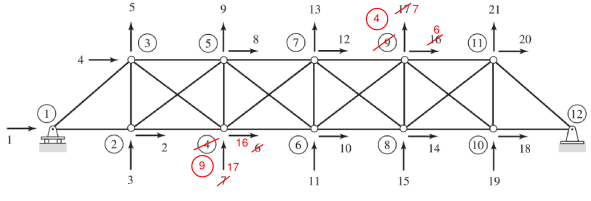


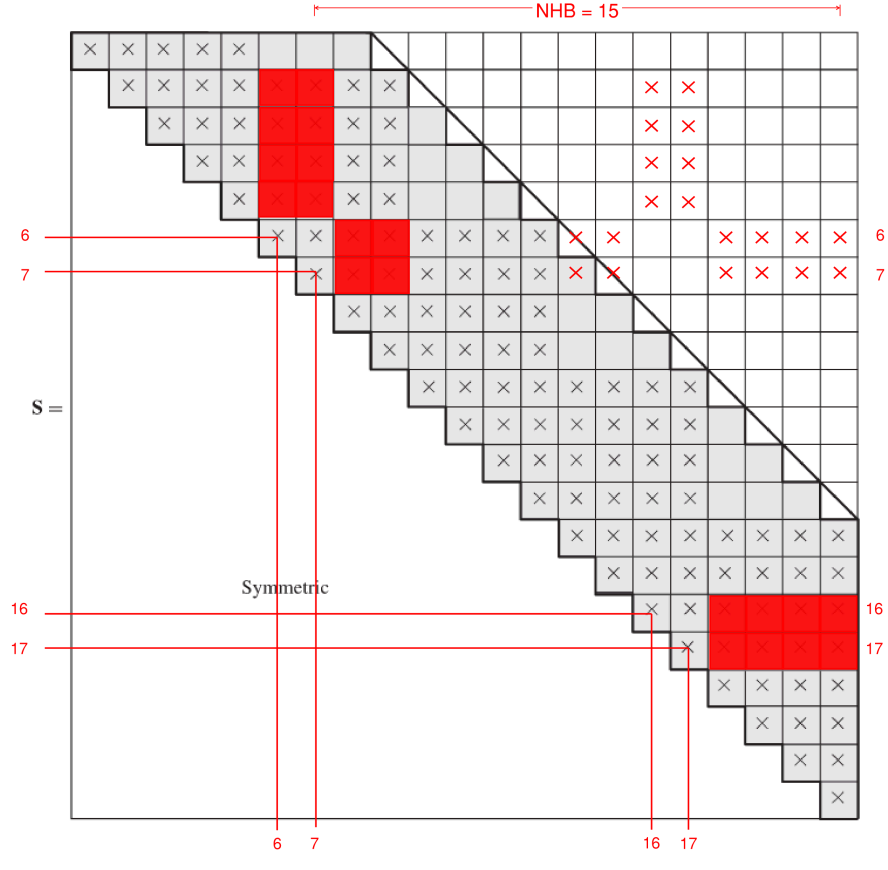
Example 2: A poorly-numbered system

Swap Node 6 and 9, number DOFs accordingly



Half-Bandwidth Increases from 7 to 15





Timing Demo — Same system, different bandwidth

Goal: compare solve times for the **same underlying SPD system** under two different degree-of-freedom orderings.

We construct matrices with:

- identical size n
- identical physical connectivity and sparsity pattern
- but very different **effective half-bandwidths**

This isolates the computational impact of bandwidth and ordering.

Solver types used in the timing demo

We solve the **same linear system** using three strategies that make different assumptions about matrix structure.

1) Dense solve (NumPy)
`np.linalg.solve(K, B)`

- Treats $\mathbf{K}$ as fully dense (LU-type)
- Ignores sparsity, bandwidth, and symmetry

2) Sparse LU (no reordering)
`splu(K, permc_spec="NATURAL")`

- Sparse storage, original DOF ordering
- Strongly affected by bandwidth and fill-in

3) Sparse LU (with reordering)
`splu(K, permc_spec="COLAMD")`

- Applies fill-reducing reordering
- Reduces bandwidth and factorization cost

Details of sparse factorization are beyond this lecture (see Kassimali §9.9).

Switch to demo code

What this demo shows

- The two systems represent the **same physics**, differing only in DOF ordering
- Dense solvers ignore sparsity and bandwidth entirely (unaffected)
- Larger bandwidth typically leads to higher factorization cost for sparse solvers
- Sparse solvers are sensitive to ordering, but can reduce its impact via reordering

Takeaway: DOF ordering changes the algebraic structure of $\mathbf{K}$, which can strongly influence computational cost even though the physics is unchanged.