CEE6501 — Lecture 4.3

Matrix Bandwidth and Extending the DSM to 3D

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 / DOF ordering** changes bandwidth without changing physics
- Demonstrate (with timing) why exploiting structure can accelerate linear solves
- Extend the Direct Stiffness Method (DSM) from 2D to **3D trusses**
- Write the **3D truss element** stiffness in local form, the 3D transformation matrix, and the assembled global element stiffness
- State the **support constraints** needed to prevent rigid body motion in 3D

Agenda

1. Sparsity in the global stiffness matrix
2. Half-bandwidth and why node numbering matters
3. A timing demo: same size, different bandwidth
4. DSM in 3D: DOFs, rotations, and supports
5. 3D truss element matrices: $\mathbf{k}_{local}$, $\mathbf{T}$, $\mathbf{k}_{global}$

Part 1 — Sparsity and Bandwidth

> *In large structural systems, performance depends not only on how many nonzeros we have, but on where they appear in the matrix.*

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.

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.

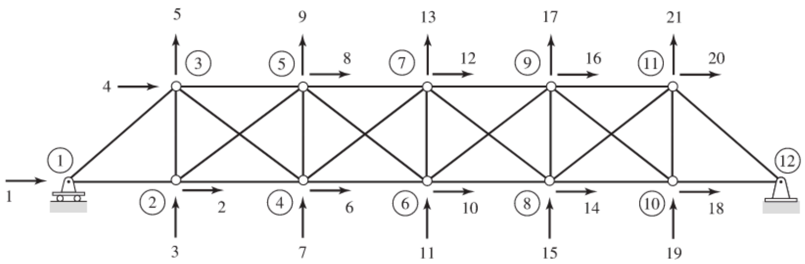
Why node numbering matters

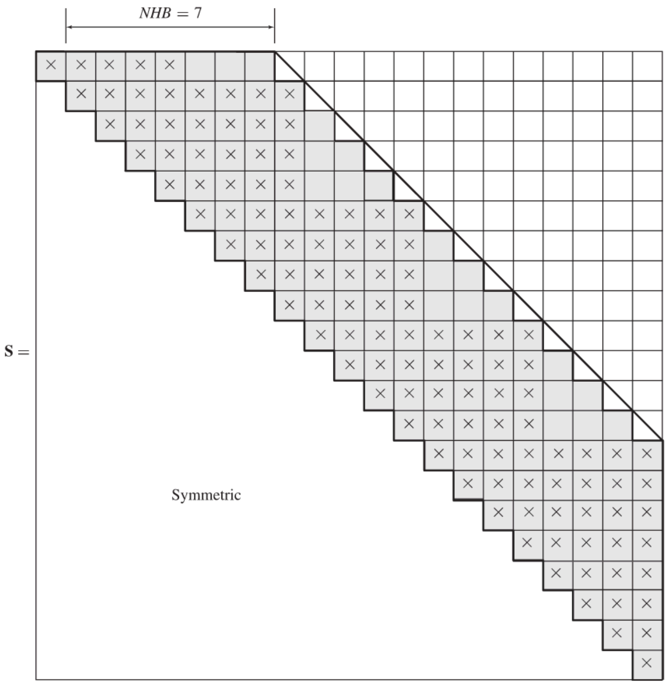
The physical structure is unchanged, but the **algebraic representation** of $\mathbf{K}$ depends on the ordering of degrees of freedom.

- 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.

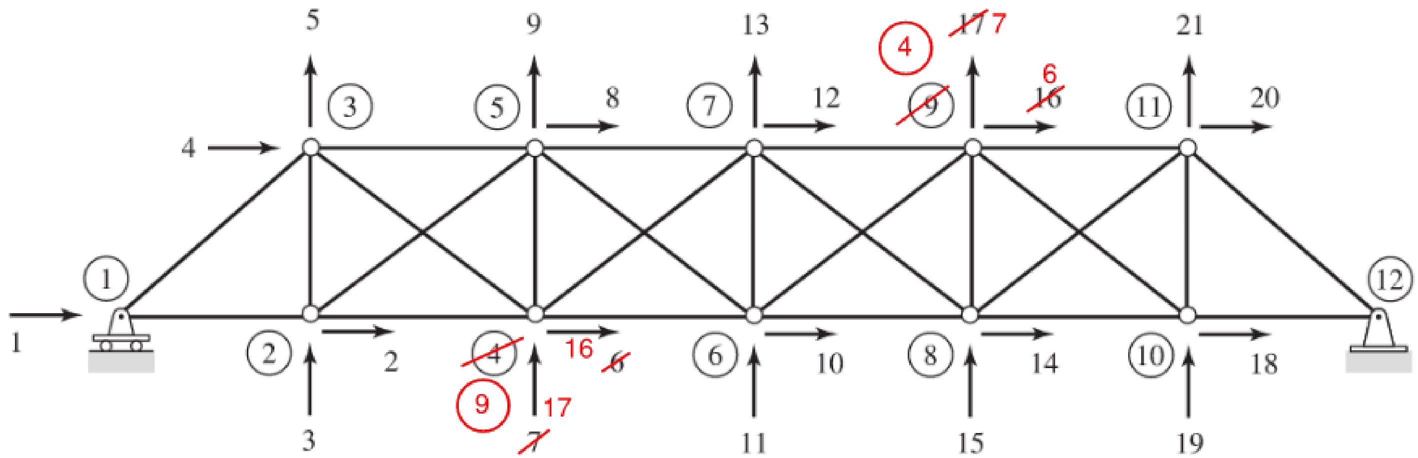
Example 1: A well-numbered system



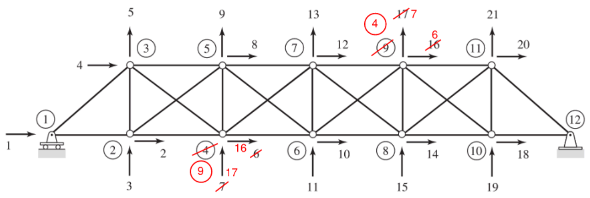


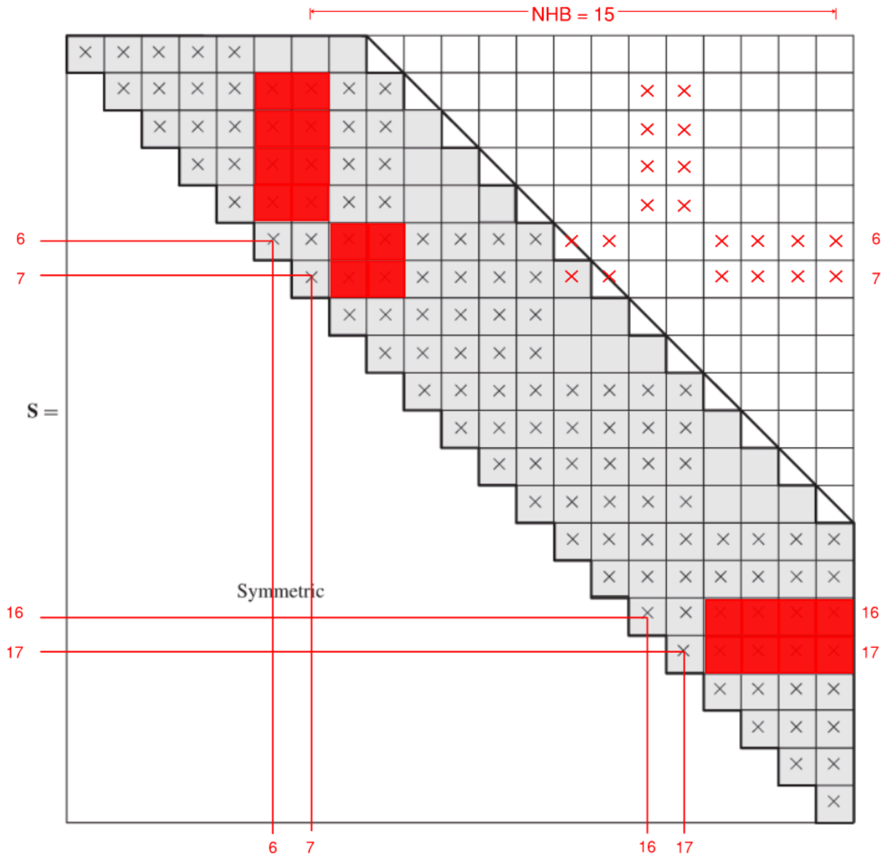
Example 2: A poorly-numbered system

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.

Part 2 — Extending the DSM to 3D Trusses

> *Same DSM workflow, but with 3 translational DOFs per node and a 3D orientation.*

Degrees of freedom in 3D

For a 3D truss node:

$$\mathbf{d}_i = \begin{bmatrix} u_{xi} & u_{yi} & u_{zi} \end{bmatrix}^T, \quad \mathbf{d}_j = \begin{bmatrix} u_{xj} & u_{yj} & u_{zj} \end{bmatrix}^T$$

Element displacement vector in global coordinates:

$$\mathbf{d}_e = \begin{bmatrix} u_{xi} & u_{yi} & u_{zi} & u_{xj} & u_{yj} & u_{zj} \end{bmatrix}^T \equiv \text{DOFs } [1, 2, 3, 4, 5, 6]$$

3D direction cosines

Let the element connect nodes i and j with coordinates (x_i, y_i, z_i) and (x_j, y_j, z_j).

$$L = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}$$

$$l = \frac{x_j - x_i}{L}, \quad m = \frac{y_j - y_i}{L}, \quad n = \frac{z_j - z_i}{L}$$

These are the direction cosines of the element axis in global coordinates.

3D Truss Element Matrices

We will write:

- the **local** element stiffness $\mathbf{k}_{local}$ (6×6)
- the **transformation** matrix $\mathbf{T}$ (6×6)
- the **global** element stiffness $\mathbf{k}_{global}$ (6×6)

Local stiffness matrix (6×6)

A truss element carries **axial force only** and therefore has stiffness only in the direction of its axis.

In a local coordinate system where the element axis is the local x' direction, only the axial DOFs are active.

Then the 6×6 local stiffness is:

$$\mathbf{k}_{local} = \frac{EA}{L} \begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Only axial coupling appears; transverse DOFs have zero stiffness in a truss model.

Transformation Matrix: Local axes and direction cosines

A 3D truss element carries **axial force only**, but we still need a consistent mapping between global and local translation components.

Let the element run from node i to node j with coordinates (x_i, y_i, z_i) and (x_j, y_j, z_j).

Define the element differences and length:

$$\Delta x = x_j - x_i, \quad \Delta y = y_j - y_i, \quad \Delta z = z_j - z_i,$$

$$L = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}.$$

The **axial direction cosines** (local x' axis) are:

$$l = \frac{\Delta x}{L}, \quad m = \frac{\Delta y}{L}, \quad n = \frac{\Delta z}{L}.$$

Why we need *two more* directions in 3D

In 2D, specifying the element axis automatically defines the perpendicular direction. In 3D, the axis (l, m, n) alone does **not** uniquely define a coordinate system: the element can still rotate about its own axis.

So we complete an orthonormal local basis:

- $\hat{\mathbf{e}}_{x'}$ along the member (given by (l, m, n))
- $\hat{\mathbf{e}}_{y'}$ transverse
- $\hat{\mathbf{e}}_{z'}$ transverse

These transverse directions do **not** add stiffness in a truss (still axial-only), but they make the transformation well-defined.

Constructing $\hat{\mathbf{e}}_{y'}$ and $\hat{\mathbf{e}}_{z'}$

Pick a reference vector $\mathbf{a}$ that is **not** parallel to the bar axis. A simple robust rule:

$$\mathbf{a} = \begin{cases} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, & |n| < 0.9 \\ \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, & |n| \geq 0.9 \quad \text{(bar nearly vertical)} \end{cases}$$

Define the local unit vectors:

$$\hat{\mathbf{e}}_{x'} = \begin{bmatrix} l \\ m \\ n \end{bmatrix}$$

$$\hat{\mathbf{e}}_{y'} = \frac{\mathbf{a} \times \hat{\mathbf{e}}_{x'}}{\|\mathbf{a} \times \hat{\mathbf{e}}_{x'}\|}$$

$$\hat{\mathbf{e}}_{z'} = \hat{\mathbf{e}}_{x'} \times \hat{\mathbf{e}}_{y'}$$

Write the transverse direction cosines as:

$$\hat{\mathbf{e}}_{y'} = \begin{bmatrix} l_y \\ m_y \\ n_y \end{bmatrix} \qquad \hat{\mathbf{e}}_{z'} = \begin{bmatrix} l_z \\ m_z \\ n_z \end{bmatrix}$$

Meaning: l_y is the x-component of the local y' axis, m_y is the y-component, etc.

3×3 rotation matrix $\mathbf{R}$ (direction cosines)

Collect the three local unit vectors (written in global components) into:

$$\mathbf{R} = \begin{bmatrix} l & m & n \\ l_y & m_y & n_y \\ l_z & m_z & n_z \end{bmatrix}$$

By construction, the basis is orthonormal, so $\mathbf{R}\mathbf{R}^T = \mathbf{I}$.

6×6 transformation matrix $\mathbf{T}$

Use global element DOFs ordered as:

$$\mathbf{d}_{global} = [u_{xi}, u_{yi}, u_{zi}, u_{xj}, u_{yj}, u_{zj}]^T$$

Use local element DOFs ordered as:

$$\mathbf{d}_{local} = [u_{x'i}, u_{y'i}, u_{z'i}, u_{x'j}, u_{y'j}, u_{z'j}]^T$$

Then:

$$\mathbf{d}_{local} = \mathbf{T}\,\mathbf{d}_{global}$$

with the block-diagonal transformation:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix}$$

Explicitly (in direction cosines):

$$\mathbf{T} = \begin{bmatrix} l & m & n & 0 & 0 & 0 \\ l_y & m_y & n_y & 0 & 0 & 0 \\ l_z & m_z & n_z & 0 & 0 & 0 \\ 0 & 0 & 0 & l & m & n \\ 0 & 0 & 0 & l_y & m_y & n_y \\ 0 & 0 & 0 & l_z & m_z & n_z \end{bmatrix}$$

How $\mathbf{T}$ connects to the 3D truss stiffness

Local (axial-only) stiffness in the element basis:

$$\mathbf{k}_{local} = \frac{EA}{L}\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Transformation to global coordinates:

$$\mathbf{k}_{global} = \mathbf{T}^T \, \mathbf{k}_{local} \, \mathbf{T}$$

Global element stiffness (6×6)

Compute:

$$\mathbf{k}_{global} = \mathbf{T}^T \, \mathbf{k}_{local} \, \mathbf{T}$$

For a 3D truss element, the result can be written directly as:

$$\mathbf{k}_{global} = \frac{EA}{L} \begin{bmatrix} l^2 & lm & ln & -l^2 & -lm & -ln \\ lm & m^2 & mn & -lm & -m^2 & -mn \\ ln & mn & n^2 & -ln & -mn & -n^2 \\ -l^2 & -lm & -ln & l^2 & lm & ln \\ -lm & -m^2 & -mn & lm & m^2 & mn \\ -ln & -mn & -n^2 & ln & mn & n^2 \end{bmatrix}$$

This is the standard 3D truss element stiffness in global coordinates. Even though $\mathbf{T}$ is a full 3D rotation, a truss remains axial-only. That is why the closed-form $\mathbf{k}_{global}$ only contains l, m, n terms.

In [8]:
```
import sympy as sp

# --- symbols ---
EA_L = 1

l, m, n = sp.symbols("l m n", real=True)
ly, my, ny = sp.symbols("l_y m_y n_y", real=True)
```

```python
lz, mz, nz = sp.symbols("l_z m_z n_z", real=True)

# --- T matrix (6x6) ---
T = sp.Matrix([
    [l,  m,  n,  0,  0,  0],
    [ly, my, ny, 0,  0,  0],
    [lz, mz, nz, 0,  0,  0],
    [0,  0,  0,  l,  m,  n],
    [0,  0,  0,  ly, my, ny],
    [0,  0,  0,  lz, mz, nz],
])

# --- local stiffness (6x6) ---
k_local = EA_L * sp.Matrix([
    [ 1, 0, 0, -1, 0, 0],
    [ 0, 0, 0,  0, 0, 0],
    [ 0, 0, 0,  0, 0, 0],
    [-1, 0, 0,  1, 0, 0],
    [ 0, 0, 0,  0, 0, 0],
    [ 0, 0, 0,  0, 0, 0],
])

# --- global stiffness via transformation ---
k_global = sp.simplify(T.T * k_local * T)

# Pretty output (in notebook)
sp.pprint(k_global)
```

$$\begin{bmatrix} l^2 & l\cdot m & l\cdot n & -l^2 & -l\cdot m & -l\cdot n \\ l\cdot m & m^2 & m\cdot n & -l\cdot m & -m^2 & -m\cdot n \\ l\cdot n & m\cdot n & n^2 & -l\cdot n & -m\cdot n & -n^2 \\ -l^2 & -l\cdot m & -l\cdot n & l^2 & l\cdot m & l\cdot n \\ -l\cdot m & -m^2 & -m\cdot n & l\cdot m & m^2 & m\cdot n \\ -l\cdot n & -m\cdot n & -n^2 & l\cdot n & m\cdot n & n^2 \end{bmatrix}$$

Supports and stability in 3D

A free 3D structure has **3 rigid body translations**.

For a 3D **truss** model (translations only), you must prevent rigid body motion by constraining enough nodal translations.

Typical sufficient constraint patterns include (examples):

- Fix all three translations at one node, and two at a second node, and one at a third node (if geometry allows)
- Or use support conditions consistent with the physical supports (pins/rollers) but ensuring no global drift

If constraints are insufficient, $\mathbf{K}$ is singular → the model has a mechanism.