Mechanics of Structures (CE130N)
Lab 6

1 Objective

The objective of this lab is to implement the techniques introduced to analyze statically determinate and indeterminate truss and frame structures. When the size of the structure is small, i.e., the number of unknown quantities is small, one can solve the problems by hand, but as the size of the problem increases, the complexity of hand solutions grows exponentially. The systematic method introduced allows one to easily analyze large structures.

In this lab you will be asked to write some functions and procedures which will allow you to complete a program which analyzes general truss and frame structures. The steps you will follow can be roughly organized as,

1. **3D Truss structures**
   - Extend your program to analyze 3D truss structures
2. **2D Frame structures**
   - Understand the data structure which describes the problem,
   - Solve for the the displacements and forces of frame structures.

2 Analysis of truss and beam structures

2.1 Definition of some variables

Let us define the following quantities,

$ndim$	: Number of dimensions
$ndof$	: Number of degrees of freedom per node
num_ev	: Number of variables per element
nnp	: Number of node points
nel	: Number of elements

Some examples for the values of $ndim$, $ndof$, and num_ev for some problems are summarized in Table 1. These along with the nnp and nel define the size of the vectors and matrices involved in the problem.

Table 1: Values of $ndim$, $ndof$, num_ev for some problems

	2D trusses	3D trusses	2D frames
$ndim$	2	3	2
$ndof$	2 (x, y translations)	3 (x, y, z translations)	3 (x, y translations, θ rotation)
num_ev	1 (R tension-compression force)	1 (R tension-compression force)	3 (R tension-compression force, M_A, M_B moment at two ends)

2.2 Governing equation and boundary conditions

The governing equations for a general truss or frame structure are the following,

- Equilibrium

$$\mathbf{F} = \mathbf{A}^T\mathbf{R} \tag{1}$$

  Here $\mathbf{F} \in \mathbb{R}^{nnp\cdot ndof\times 1}$ is the vector of nodal forces, $\mathbf{R} \in \mathbb{R}^{nel\cdot num\_ev\times 1}$ is the vector of element forces, and $\mathbf{A} \in \mathbb{R}^{nel\cdot num\_ev\times nnp\cdot ndof}$ is the compatibility matrix.

- Effective Kinematics

$$\mathbf{V} = \mathbf{A}\mathbf{u} \tag{2}$$

  Here $\mathbf{u} \in \mathbb{R}^{nnp\cdot ndof\times 1}$ is the vector of nodal displacements, $\mathbf{V} \in \mathbb{R}^{nel\cdot num\_ev\times 1}$ is the vector of element deformations. In the case of a truss, $\mathbf{V} = \mathbf{\Delta L}$.

- Effective Constitutive Relation

$$\mathbf{R} = \mathbf{K}_s\mathbf{V} \tag{3}$$

  Here $\mathbf{K}_s \in \mathbb{R}^{nel\cdot num\_ev\times nel\cdot num\_ev}$ is the matrix relating the element forces and element deformations. In the case of a truss, $\mathbf{K}_s = [AE/L]$.

These three relations can be combined into a single equation representing equilibrium in terms of the vector of displacements $\mathbf{u}$,

$$\boxed{\mathbf{F} = \mathbf{K}\mathbf{u},} \tag{4}$$

where,

$$\mathbf{K} \;:=\; \mathbf{A}^T\mathbf{K}_s\mathbf{A}\;.$$

One can apply boundary conditions to these equations by identifying the (F)REE degrees-of-freedom idf and (D)ISPLACEMENT known degrees-of-freedom idd to form the system of equations,

$$\left[\begin{array}{c}\mathbf{F}_f\\ \mathbf{F}_d\end{array}\right] \;=\; \left[\begin{array}{cc}\mathbf{K}_{ff} & \mathbf{K}_{fd}\\ \mathbf{K}_{df} & \mathbf{K}_{dd}\end{array}\right]\left[\begin{array}{c}\mathbf{u}_f\\ \mathbf{u}_d\end{array}\right].$$

Here $\mathbf{u}_f$ contains the displacement degrees-of-freedom corresponding to idf and $\mathbf{u}_d$ to those corresponding to idd. Employing the fact that $\mathbf{u}_d$ is known, we can solve for the $\mathbf{u}_f$ and $\mathbf{F}_d$ in the following two steps,

1. Solve for the $\mathbf{u}_f$. Define,

$$\mathbf{r}_f \;:=\; \mathbf{F}_f - \mathbf{K}_{fd}\mathbf{u}_d = \mathbf{K}_{ff}\mathbf{u}_f\;,$$

   and solve,

$$\boxed{\mathbf{u}_f = \mathbf{K}_{ff}^{-1}\mathbf{r}_f\;.} \tag{5}$$

2. Evaluate $\mathbf{F}_d$,

$$\mathbf{F}_d \;=\; \mathbf{K}_{df}\mathbf{u}_f + \mathbf{K}_{dd}\mathbf{u}_d\;,$$

   or by,

$$\mathbf{F} \;=\; \mathbf{K}\mathbf{u}\;,$$

   since now $\mathbf{u}$ is solved and a known quantity.

3 Numerical implementation

3.1 Solution procedure

A program to solve for the displacements and forces of truss and frame structures can be summarized as:

1. Determine the geometry and boundary condition (loading and supports).
2. Form the matrices $\mathbf{A}$, $\mathbf{K}_s$, and compute $\mathbf{K} = \mathbf{A}^T\mathbf{K}_s\mathbf{A}$.
3. Identify the (F)ree degrees-of-freedom and known (D)isplacement degrees-of-freedom, to extract $\mathbf{K}_{ff}$ and form $\mathbf{r}_f$.
4. Solve for the displacements $\mathbf{u}_f$ through the equation $\mathbf{r}_f = \mathbf{K}_{ff}\mathbf{u}_f$. Consecutively compute other desired quantities such as bar forces $\mathbf{R}$.

3.2 Forming the compatibility matrix A

From the previous sections it is clear that the size of the matrix $\mathbf{A}$ depends on the number of nodes nnp, number of elements nel, number of degrees-of-freedom per node $ndof$, and the number of variables per element num_ev. The important aspect of $\mathbf{A}$ is that it has a nice block structure, i.e., $\mathbf{A}$ looks like the following,

$$\mathbf{A} \rightarrow \overbrace{\qquad}^{nnp \text{ blocks}} \quad nel \text{ blocks} \left\{ \begin{bmatrix} \square & \square & \cdots & \square \\ \square & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \square & \cdots & \cdots & \square \end{bmatrix} \right. .$$

In this case each of the boxes $\square$ are num_ev-by-$ndof$ sized matrices. This implies that the size of the matrix $\square$ depends on the problem one is treating. In the case of a 2D truss, the size is 1-by-2. The actual size of $\mathbf{A}$ is $(nel \cdot num\_ev)$-by-$(nnp \cdot ndof)$.

In our numerical implementation, we have a function,

```
[Ae] = assembleAe_XXX(mesh,ie);
```

which returns the `ie`th element contribution `Ae` to the compatibility matrix $\mathbf{A}$. The size of `Ae` here is num_ev-by-$2ndof$. This means that it has the structure,

$$\texttt{Ae} = \begin{bmatrix} \square & \blacksquare \end{bmatrix}.$$

Since this is the `ie`th element contribution, $\square$ and $\blacksquare$ both go in the `ie`th block row of $\mathbf{A}$. The `ie`th block row corresponds to,

$$\texttt{ie} \text{ th block row} \rightarrow \begin{cases} (\texttt{ie}-1) \times \texttt{num\_ev} + 1 \\ \vdots \\ (\texttt{ie}-1) \times \texttt{num\_ev} + \texttt{num\_ev} \end{cases},$$

and thus it will actually go into these rows of $\mathbf{A}$. Let us assume that the element is connected to node `ia` at end A and node `ib` at end B. Then $\square$ will go in the `ia`th block column and $\blacksquare$ in the `ib`th block column. The `ia`th block column corresponds to,

$$\texttt{ia} \text{ th block column} \rightarrow \begin{cases} (\texttt{ia}-1) \times \texttt{ndof} + 1 \\ \vdots \\ (\texttt{ia}-1) \times \texttt{ndof} + \texttt{ndof} \end{cases},$$

and thus $\square$ will actually go into these columns of $\mathbf{A}$. The `ib`th block column corresponds to,

$$\texttt{ib}\text{ th block column} \rightarrow \begin{cases} (\texttt{ib}-1)\times\texttt{ndof}+1 \\ \vdots \\ (\texttt{ib}-1)\times\texttt{ndof}+\texttt{ndof} \end{cases},$$

and thus $\blacksquare$ will actually go into these columns of $\mathbf{A}$.

3.3 Forming the effective stiffness matrix $\mathbf{K}_s$

From the previous sections it is clear that the size of the matrix $\mathbf{K}_s$ depends on the number of elements nel and the number of variables per element num_ev. The important aspect of $\mathbf{K}_s$ is that it has a nice block structure, i.e., $\mathbf{K}_s$ looks like the following,

$$\mathbf{K}_s \rightarrow \overbrace{\qquad}^{nel\text{ blocks}} \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} nel\text{ blocks} \begin{bmatrix} \square & \square & \cdots & \square \\ \square & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \square & \cdots & \cdots & \square \end{bmatrix}.$$

In this case each of the boxes $\square$ are num_ev-by-num_ev sized matrices. This implies that the size of the matrix $\square$ depends on the problem one is treating. In the case of a 2D truss, the size is 1-by-1, i.e., $\mathbf{K}_s$ is a diagonal matrix. The actual size of $\mathbf{K}_s$ is $(nel \cdot num\_ev)$-by-$(nel \cdot num\_ev)$.

In our numerical implementation, we have a function,

```
[Ae] = assembleKse_XXX(mesh,ie);
```

which returns the `ie`th element contribution `Kse` to the matrix $\mathbf{K}_s$. The size of `Kse` here is num_ev-by-num_ev. This means that it has the structure,

$$\texttt{Kse} = [\square].$$

Since this is the `ie`th element contribution, $\square$ goes into the `ie`th block row and column of $\mathbf{K}_s$. The `ie`th block row and column corresponds to,

$$\texttt{ie}\text{ th block row} \rightarrow \begin{cases} (\texttt{ie}-1)\times\texttt{num\_ev}+1 \\ \vdots \\ (\texttt{ie}-1)\times\texttt{num\_ev}+\texttt{num\_ev} \end{cases},$$

and thus it will actually go into these rows and columns of $\mathbf{K}_s$.

4 3D Truss structures

4.1 Data structure

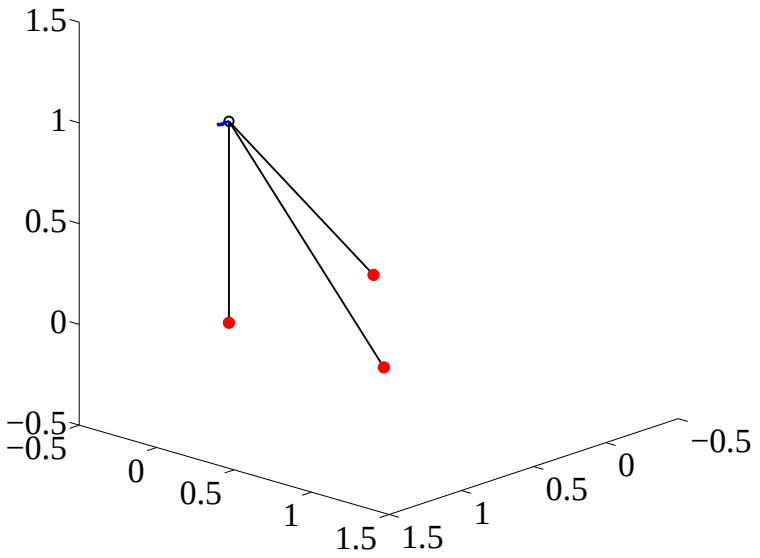

Figure 1: 3D truss example

To solve the truss problem numerically, one needs more than a picture. One requires an abstract representation of the truss structure along with the boundary conditions. Various representations are possible and below we present the data structure we will be using in our implementation. The data structure is for the truss shown in Figure 1. This is identical to the problem you have treated in the lecture.

```
clear mesh;

% -- Specify the problem
mesh.ndim  = 3;       % -- Dimension of problem
mesh.ndof  = 3;       % -- Degrees of freedom per node
mesh.num_ev= 1;       % -- Number of element variables
mesh.etype = 'truss'; % -- Type of element

% -- Node to coordinate array
%
%        mesh.node  -- An array of size (ndim)-by-(number of nodes)
%
%    For the 3D case,
%
%        mesh.node =     [node1.x, node2.x,...]
%                        [node1.y, node2.y,...]
%                        [node1.z, node2.z,...]
%
mesh.node = [0, 1, 1, 1;
             0, 0, 1, 0;
             0, 0, 0, 1;];

% -- Element connectivity array
%
%        mesh.conn  -- An array of size
```

```
%                      (number of connections)-by-(number of elements)
%
%    For the 'truss' case,
%
%        mesh.conn =  [element1.n1, element2.n1,...]
%                     [element1.n2, element2.n2,...]
%
mesh.conn = [1, 2, 3;
             4, 4, 4;];

% -- Material properties
%
%        mesh.mat         -- An array of structures of
%                            size (number of elements)
%
%    The material properties for 'i'th element number
%
%        mesh.mat{i}.E    -- Youngs modulus
%        mesh.mat{i}.A    -- Cross-sectional area
%
mat1.E     = 1e3;
mat1.A     = 1;
mesh.mat{1}= mat1;
mesh.mat{2}= mat1;
mesh.mat{3}= mat1;

% -- Node to boundary condition code array
%
%        mesh.bc -- An array of size
%                   (ndof)-by-(number of nodes)
%
%    The entries of this array take either a value of 0 or 1.
%    For the i-th degree of freedom for the j-th node,
%
%        mesh.bc(i,j) = 0     -- Implies Known Force        BC
%        mesh.bc(i,j) = 1     -- Implies Known Displacement BC
%
mesh.bc     = zeros(mesh.ndof,size(mesh.node,2));
                      % -- Initally assume all Known Force BC
mesh.bc(:,1) = [1;
                1;
                1;];
mesh.bc(:,2) = [1;
                1;
                1;];
mesh.bc(:,3) = [1;
                1;
                1;];

% -- Nodal displacement value array
%
%        mesh.u -- An array of size
%                  (ndof)-by-(number of nodes)
%
%    The entries of this array store the value of the nodal displacements
%    The displacement in the direction of the i-th degree of freedom
%                                      for the j-th node
%    is  mesh.u(i,j)
%
%    One must preset the nodal displacement values for
%    the KNOWN DISPLACEMENT BCs
%
mesh.u      = zeros(mesh.ndof,size(mesh.node,2));
mesh.u(:,1) = [0.0;
               0.0;
```

```
               0.0;];
mesh.u(:,2) = [0.0;
               0.0;
               0.0;];
mesh.u(:,3) = [0.0;
               0.0;
               0.0;];

% -- Nodal force value array
%
%        mesh.f -- An array of size
%                  (ndof)-by-(number of nodes)
%
%    The entries of this array store the value of the nodal forces
%    The force in the direction of the i-th degree of freedom
%                               for the j-th node
%    is  mesh.f(i,j)
%
%    One must preset the nodal force values for
%    the KNOWN FORCE BCs
%
mesh.f      = zeros(mesh.ndof,size(mesh.node,2));
mesh.f(:,4) = [1.0;
               0.0;
               0.0;];
```

5 Frame structures

5.1 Governing equation

Let us define the following quantities,

$ndof$: Number of degrees-of-freedom per node, for 2D frames(2)
nnp : Number of node points
nel : Number of elements (beams)

The governing equations for a frame structure are the following,

- Equilibrium

$$\mathbf{F} = \mathbf{A}^T\mathbf{R} \tag{6}$$

Here $\mathbf{F} \in \mathbb{R}^{ndof \cdot nnp \times 1}$ is the vector of nodal forces, $\mathbf{R} \in \mathbb{R}^{3nel \times 1}$ is the vector of beam forces, and $\mathbf{A} \in \mathbb{R}^{3nel \times ndof \cdot nnp}$ is the compatibility matrix.

- Kinematics

$$\mathbf{V} = \mathbf{A}\mathbf{u}, \tag{7}$$

Here $\mathbf{u} \in \mathbb{R}^{ndof \cdot nnp \times 1}$ is the vector of nodal displacements, $\mathbf{V} \in \mathbb{R}^{3nel \times 1}$ is the vector of beam deformations.

- Effective constitutive relation

$$\mathbf{R} = \mathbf{K}_s\mathbf{V} \tag{8}$$

Here $\mathbf{K}_s \in \mathbb{R}^{3nel \times 3nel}$ is the block diagonal matrix of beam stiffnesses.

For the case of this 2D-frame structure, one has an additional degree of freedom at each node compared to truss structures. This degree-of-freedom is the rotational degree of freedom $u_{i,\theta}$. Thus $\mathbf{u}$ has the following structure,

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{nnp} \end{bmatrix}$$

where the contribution from each node is,

$$\mathbf{u}_i = \begin{bmatrix} u_{i,x} \\ u_{i,y} \\ u_{i,\theta} \end{bmatrix}.$$

The vector of nodal forces $\mathbf{F}$ will have the same structure,

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_1 \\ \vdots \\ \mathbf{F}_{nnp} \end{bmatrix}$$

where the contribution from each node is,

$$\mathbf{F}_i = \begin{bmatrix} F_{i,x} \\ F_{i,y} \\ F_{i,M} \end{bmatrix}$$

and $F_{i,M}$ is the applied moment at node i.

The beam also has two more additional variables to define its deformation compared to the bar. These are the rotations at the two ends, $\theta_{i,A}$ and $\theta_{i,B}$. Thus $\mathbf{V}$ has the following structure,

$$\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_{nel} \end{bmatrix}$$

where the deformations of each beam are,

$$\mathbf{v}_i = \begin{bmatrix} \Delta L_i \\ \theta_{i,A} \\ \theta_{i,B} \end{bmatrix}.$$

One can also apply moments at the two ends of the beam which add two more variables to the element forces, $M_{i,A}$ and $M_{i,B}$. The vector of beam forces $\mathbf{R}$ will have the structure,

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_1 \\ \vdots \\ \mathbf{R}_{nel} \end{bmatrix}$$

where the forces in each beam are,

$$\mathbf{R}_i = \begin{bmatrix} R_i \\ M_{i,A} \\ M_{i,B} \end{bmatrix}.$$

The relationship between the beam forces $\mathbf{R}_i$ and beam deformations $\mathbf{v}_i$ for beam i, is defined as,

$$\begin{aligned} \mathbf{R}_i &= \mathbf{K}_{s,i}\mathbf{v}_i, \\ \mathbf{K}_{s,i} &:= \begin{bmatrix} \frac{EA}{L} & 0 & 0 \\ 0 & \frac{4EI}{L} & \frac{2EI}{L} \\ 0 & \frac{2EI}{L} & \frac{4EI}{L} \end{bmatrix} \end{aligned}$$

Thus the matrix $\mathbf{K}_s$ in the effective constitutive relation will have the following block diagonal structure,

$$\mathbf{K}_s = \begin{bmatrix} \mathbf{K}_{s,1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{K}_{s,2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{K}_{s,nel} \end{bmatrix}$$

5.2 Data structure

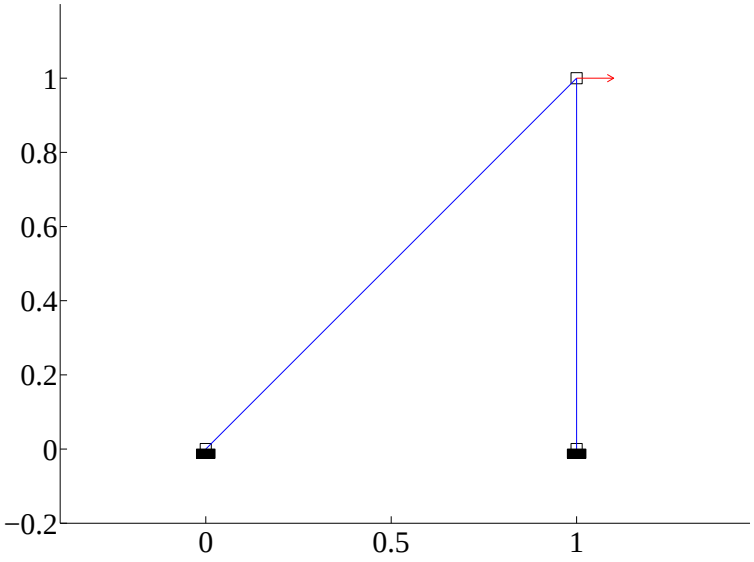

Figure 2: Frame example

To solve the frame problem numerically, one needs more than a picture. One requires an abstract representation of the frame structure along with the boundary conditions. Various representations are possible and below we present the data structure we will be using in our implementation. The data structure is for the frame shown in Figure 2. This is identical to the problem you have treated in the lecture.

```
clear mesh;

% -- Specify the problem
mesh.ndim  = 2;       % -- Dimension of problem
mesh.ndof  = 3;       % -- Degrees of freedom per node
mesh.num_ev= 3;       % -- Number of element variables
mesh.etype = 'beam';  % -- Type of element

% -- Node to coordinate array
%
%        mesh.node  -- An array of size (ndim)-by-(number of nodes)
%
%    For the 2D case,
%
%        mesh.node =     [node1.x, node2.x,...]
%                        [node1.y, node2,y,...]
%
mesh.node = [0, 1, 1;
             0, 0, 1;];

% -- Element connectivity array
%
%        mesh.conn  -- An array of size
%                      (number of connections)-by-(number of elements)
%
%    For the 'beam'  case,
%
%        mesh.conn =  [element1.n1, element2.n1,...]
%                     [element1.n2, element2.n2,...]
%
```

```
mesh.conn = [1, 3;
             3, 2;];

% -- Material properties
%
%        mesh.mat         -- An array of structures of
%                            size (number of elements)
%
%    The material properties for 'i'th element number
%
%        mesh.mat{i}.E    -- Youngs modulus
%        mesh.mat{i}.A    -- Cross-sectional area
%
mat1.E     = 1e3;
mat1.A     = 1;
mat1.I     = 1;
mesh.mat{1}= mat1;
mesh.mat{2}= mat1;

% -- Node to boundary condition code array
%
%        mesh.bc -- An array of size
%                   (ndof)-by-(number of nodes)
%
%    The entries of this array take either a value of 0 or 1.
%    For the i-th degree of freedom for the j-th node,
%
%        mesh.bc(i,j) = 0     -- Implies Known Force        BC
%        mesh.bc(i,j) = 1     -- Implies Known Displacement BC
%
mesh.bc     = zeros(mesh.ndof,size(mesh.node,2));
                      % -- Initally assume all Known Force BC
mesh.bc(:,1) = [1;
                1;
                1;];
mesh.bc(:,2) = [1;
                1;
                1;];

% -- Nodal displacement value array
%
%        mesh.u -- An array of size
%                  (ndof)-by-(number of nodes)
%
%    The entries of this array store the value of the nodal displacements
%    The displacement in the direction of the i-th degree of freedom
%                                       for the j-th node
%    is  mesh.u(i,j)
%
%    One must preset the nodal displacement values for
%    the KNOWN DISPLACEMENT BCs
%
mesh.u      = zeros(mesh.ndof,size(mesh.node,2));
mesh.u(:,1) = [0.0;
               0.0;
               0.0];
mesh.u(:,2) = [0.0;
               0.0;
               0.0];

% -- Nodal force value array
%
%        mesh.f -- An array of size
%                  (ndof)-by-(number of nodes)
%
```

```
%    The entries of this array store the value of the nodal forces
%    The force in the direction of the i-th degree of freedom
%                              for the j-th node
%    is  mesh.f(i,j)
%
%    One must preset the nodal force values for
%    the KNOWN FORCE BCs
%
mesh.f      = zeros(mesh.ndof,size(mesh.node,2));
mesh.f(:,3) = [1.0;
               0.0;
               0.0;];
```

5.3 Element-by-element assembly of the compatibility matrix A

It has been introduced in the lecture that the $\mathbf{A}$ can be formed systematically. Consider the compatibility matrix of Figure 2,

$$\mathbf{A} = \begin{bmatrix} \mathbf{E}_{31a}^T & \mathbf{0} & \mathbf{E}_{13b}^T \\ \mathbf{0} & \mathbf{E}_{32b}^T & \mathbf{E}_{23a}^T \end{bmatrix},$$

which relates the beam deformations $\mathbf{V}$, with the nodal displacements $\mathbf{u}$,

$$\begin{aligned} \mathbf{V} &= \mathbf{A}\mathbf{u}, \\ \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{bmatrix} &= \begin{bmatrix} \mathbf{E}_{31a}^T & \mathbf{0} & \mathbf{E}_{13b}^T \\ \mathbf{0} & \mathbf{E}_{32b}^T & \mathbf{E}_{23a}^T \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \end{bmatrix}. \end{aligned}$$

Here the matrices $\mathbf{E}_{ABa}$ and $\mathbf{E}_{ABb}$ are 3-by-3 matrices which have the form,

$$\begin{aligned} \mathbf{E}_{ABa} &:= \begin{bmatrix} \mathbf{e}_{AB} & \frac{\mathbf{n}_{AB}}{L} & \frac{\mathbf{n}_{AB}}{L} \\ 0 & 1 & 0 \end{bmatrix}, \\ \mathbf{E}_{ABb} &:= \begin{bmatrix} \mathbf{e}_{AB} & \frac{\mathbf{n}_{AB}}{L} & \frac{\mathbf{n}_{AB}}{L} \\ 0 & 0 & 1 \end{bmatrix}, \end{aligned}$$

and $\mathbf{e}_{AB}$ is the unit vector point from node A to node B and $\mathbf{n}_{AB}$ is the unit vector orthogonal to $\mathbf{e}_{AB}$ obtained by rotating $\mathbf{e}_{AB}$ clockwise by 90 degrees. The rows of $\mathbf{A}$ correspond to the beams, and the columns correspond to the nodes. Each beam only links to two nodes, so there are only two entries per row. In order to assemble $\mathbf{A}$, one can iterate through the beams starting from beam 1 to beam nel, inserting one row at a time.

In our implementation, there is a function,

```
function [Ae] = assembleAe_beam(mesh,ie)
```

which given the element number ie and `mesh` data structure, returns $\mathbf{A}_e$, the element contribution to the compatibility matrix, defined as

$$\mathbf{v}_{ie} = \mathbf{A}_e \begin{bmatrix} \mathbf{u}_A \\ \mathbf{u}_B \end{bmatrix},$$

where,

$$\begin{aligned} \mathbf{A}_e &:= \left[\mathbf{E}_{BAa}^T, \mathbf{E}_{ABb}^T\right], \\ \mathbf{v}_{ie} &:= \begin{bmatrix} \Delta L_{ie} \\ \theta_{ie,A} \\ \theta_{ie,B} \end{bmatrix}, \\ \mathbf{u}_A &:= \begin{bmatrix} u_{A,x} \\ u_{A,y} \\ u_{A,\theta} \end{bmatrix}, \end{aligned}$$

Here it is assumed that the element connects to nodes A and B. Once the $\mathbf{A}_e$ is computed, it can be inserted into the correct location of the compatibility matrix $\mathbf{A}$ using the information from `mesh.conn`. This is done in the function,

```
function [A] = assembleA(mesh)
```

REMARK: This procedure is identical to the procedure for constructing the compatibility matrix for the truss structure where the compatibility matrix has the form,

$$\mathbf{A} = \begin{bmatrix} \mathbf{e}_{31}^T & \mathbf{0} & \mathbf{e}_{13}^T \\ \mathbf{0} & \mathbf{e}_{32}^T & \mathbf{e}_{23}^T \end{bmatrix}.$$

In this case the $\mathbf{e}_{AB}$'s are 2-by-1 matrices, i.e., vectors.

6 Exercise

6.1 Download updates for the *truss* program

1. Go to the directory `ce130n/week4/lab/matlab/truss/core` directory and rename the following files.

   `assembleA.m` → `assembleA_week5.m`
   `solvemesh.m` → `solvemesh_week5.m`

2. Download the file `truss_update_week6.zip` into your `ce130n/week4/lab/matlab/` directory and unzip it.

3. Copy the files in the directory `truss_update_week6` into the appropriate locations in your `ce130n/week4/lab/matlab/truss` directory, overwriting some previous files.

YOU MUST RUN THE FILE init.m EVERYTIME YOU START UP MATLAB.

6.2 Modify the *truss* program for 3D analysis

1. Load the sample data structure and plot the 3D truss structure.

   ```
   >> mesh_data_truss_example3d;
   >> plotmesh(mesh);
   ```

   The file `mesh_data_truss_example3d.m` contains the data structure for this truss stored in the variable `mesh`, and the function `plotmesh.m` plots the data structure `mesh`.

2. Make sure the function
   `ce130n/week4/lab/matlab/truss/element/assembleAe_truss.m`
   works for the 3D truss case. You may have to modify the way you compute the unit vector so that it applies for both the 2D and 3D case.

   To test the function for the 3D case, one must first load the mesh structure for the problem and then run the function,

   ```
   >> mesh_data_truss_example3d;
   >> [Ae] = assembleAe_truss(mesh,1);
   ```

   Make sure the function still works for the 2D case by running,

   ```
   >> mesh_data_truss_example;
   >> [Ae] = assembleAe_truss(mesh,1);
   ```

3. Complete the function
`ce130n/week4/lab/matlab/truss/core/determine_rcind.m`
which computes the row and column indices of the compatibility matrix $\mathbf{A}$ into which the element compatibility matrix $\mathbf{A}_e$ is inserted.

   To test the function you can run for example,

```
>> ie = 3;
>> ia = 2;
>> ib = 5;
>> num_ev = 2;
>> ndof   = 3;
>> [rid,cidA,cidB] = determine_rcind(num_ev,ndof,ie,ia,ib);
```

   For this case, one should obtain the results,

$$\begin{aligned} \texttt{rid} &= [5, 6]; \\ \texttt{cidA} &= [4, 5, 6]; \\ \texttt{cidB} &= [13, 14, 15]; \end{aligned}$$

4. Complete the function
`ce130n/week4/lab/matlab/truss/core/assembleA.m`
This version will implement the function you have constructed above `determine_rcind.m` and will be more general in its applicability. You will be able to form the $\mathbf{A}$ matrices for 2D/3D trusses and 2D frames.

   To test the function one must first load the mesh structure for the problem and then run the function,

```
>> mesh_data_truss_example3d;
>> solvemesh;
>> plotdefo(mesh,1e1);
```

   The correct displacement for the top node is $(0.0038, -0.0010, -0.0010)$.

   CHECKPOINT: Show the plot for the deformed structures. Explain what the reaction forces, bar forces, and bar stresses are and how you can obtain them. Compare this 3D analysis with the 2D case that you have already analyzed in the `mesh_data_truss_example.m`. Why does the top node have a negative displacement in the y direction? Why is it not zero?

6.3 Analyze frame structures

1. Load, plot, and solve the example frame structure shown in Figure 2.

```
>> mesh_data_frame_example;
>> plotmesh(mesh);
>> solvemesh(mesh);
>> plotdefo(mesh,1e3);
```

2. Construct the data structure for the irregular frame structure shown in Figure 3. Then load, plot, and solve for the displacements and forces. The applied force at the middle node is $(0, -25, 0)$ and at the top right node is $(20, 0, 0)$.

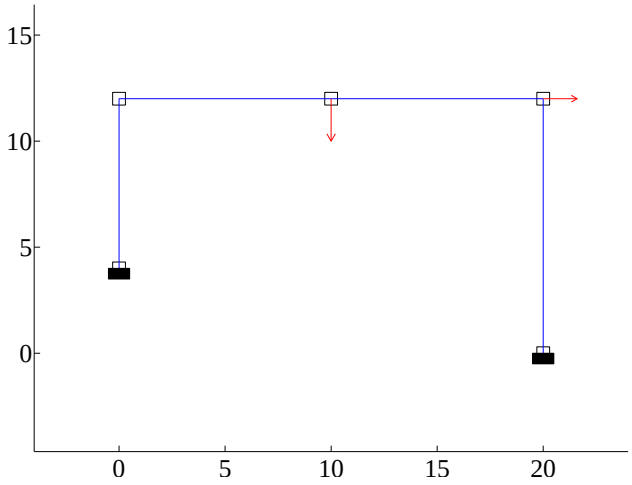

Figure 3: Irregular frame structure

CHECKPOINT: What are the displacements at the nodes? What are the reaction forces at the supports? What are the beam forces (axial force, moments, shear force)? How can one obtain the element compatibility matrix A_e for one of the beams in the frame structure?

The configuration of the frame structure `mesh_data_frame_example.m` is identical to the `mesh_data_truss_example.m`. Try to relate the results you obtain from the two structures. (HINT: Make the I for the frame example structure smaller and observe how the displacement at the top node changes).