

# Analysis of planar systems of trusses and frames

Course of Spacecraft Structures  
AY 2017/2018

Riccardo Vescovini

Politecnico di Milano, Department of Aerospace Science and Technology

# Introduction

- The main features of a simple program for the analysis of systems of frames and trusses are illustrated
- The implementation is based upon the use of the stiffness matrices of trusses and beams (obtained using the PCVW)
- Shear deformability is not accounted for (slender beams are considered)
- Any kind of planar 2D system of beams and trusses can be analyzed

# Overview of the relevant equations

- Stiffness matrix of a beam (in the local reference system)

$$\mathbf{k} = \begin{bmatrix} \frac{EA}{l} & 0 & 0 & -\frac{EA}{l} & 0 & 0 \\ 0 & 12\frac{EJ}{l^3} & 6\frac{EJ}{l^2} & 0 & -12\frac{EJ}{l^3} & 6\frac{EJ}{l^2} \\ 0 & 6\frac{EJ}{l^2} & 4\frac{EJ}{l} & 0 & -6\frac{EJ}{l^2} & 2\frac{EJ}{l} \\ -\frac{EA}{l} & 0 & 0 & \frac{EA}{l} & 0 & 0 \\ 0 & -12\frac{EJ}{l^3} & -6\frac{EJ}{l^2} & 0 & 12\frac{EJ}{l^3} & -6\frac{EJ}{l^2} \\ 0 & 6\frac{EJ}{l^2} & 2\frac{EJ}{l} & 0 & -6\frac{EJ}{l^2} & 4\frac{EJ}{l} \end{bmatrix}$$

with degrees of freedom sorted as:

$$\mathbf{u}^T = \{u_1 \quad v_1 \quad \theta_1 \quad u_2 \quad v_2 \quad \theta_2\}$$

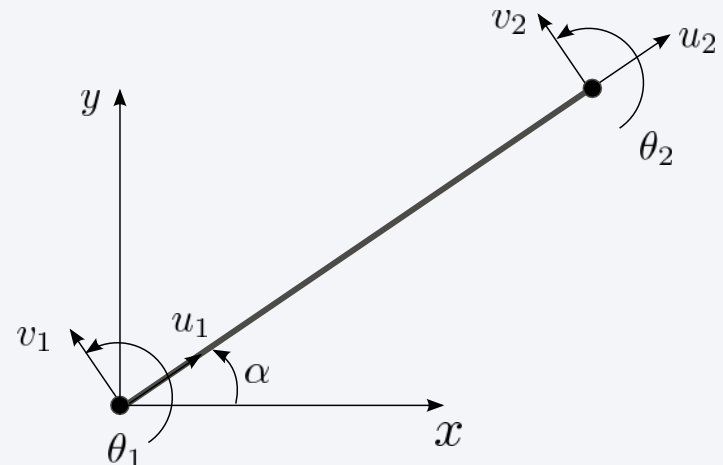
- Transformation from global to local system:

$$\mathbf{u} = \mathbf{T}\mathbf{U}$$

$$\mathbf{K} = \mathbf{T}^T \mathbf{k} \mathbf{T}$$

with:  $\mathbf{T} = \begin{bmatrix} c & s & 0 & 0 & 0 & 0 \\ -s & c & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & c & s & 0 \\ 0 & 0 & 0 & -s & c & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

$$\mathbf{U}^T = \{U_1 \quad V_1 \quad \theta_1 \quad U_2 \quad V_2 \quad \theta_2\}$$



# Overview of the relevant equations

- Stiffness matrix of a bar (in the local reference system)

$$\mathbf{k} = \frac{EA}{l} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

with degrees of freedom sorted as:  $\mathbf{u}^T = \{u_1 \quad u_2\}$

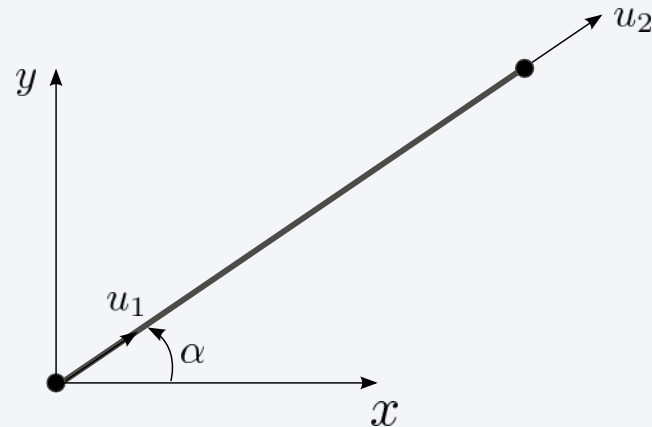
- Transformation from global to local system:

$$\mathbf{u} = \mathbf{T}\mathbf{U}$$

$$\mathbf{K} = \mathbf{T}^T \mathbf{k} \mathbf{T}$$

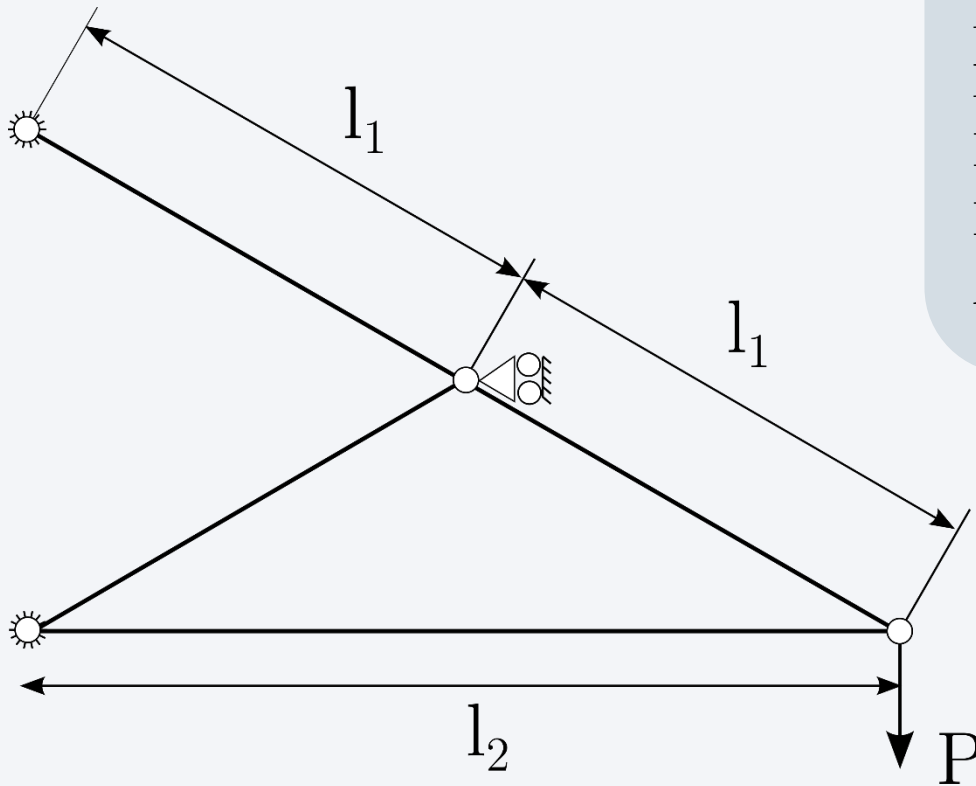
with:  $\mathbf{T} = \begin{bmatrix} c & s & 0 & 0 & 0 & 0 \\ s & c & 0 & 0 & 0 & 0 \end{bmatrix}$

$$\mathbf{U}^T = \{U_1 \quad V_1 \quad U_2 \quad V_2\}$$



# Example

- System of 4 truss elements



## Input data

$$l_1 = 1000 \text{ mm}$$

$$l_2 = 1732 \text{ mm}$$

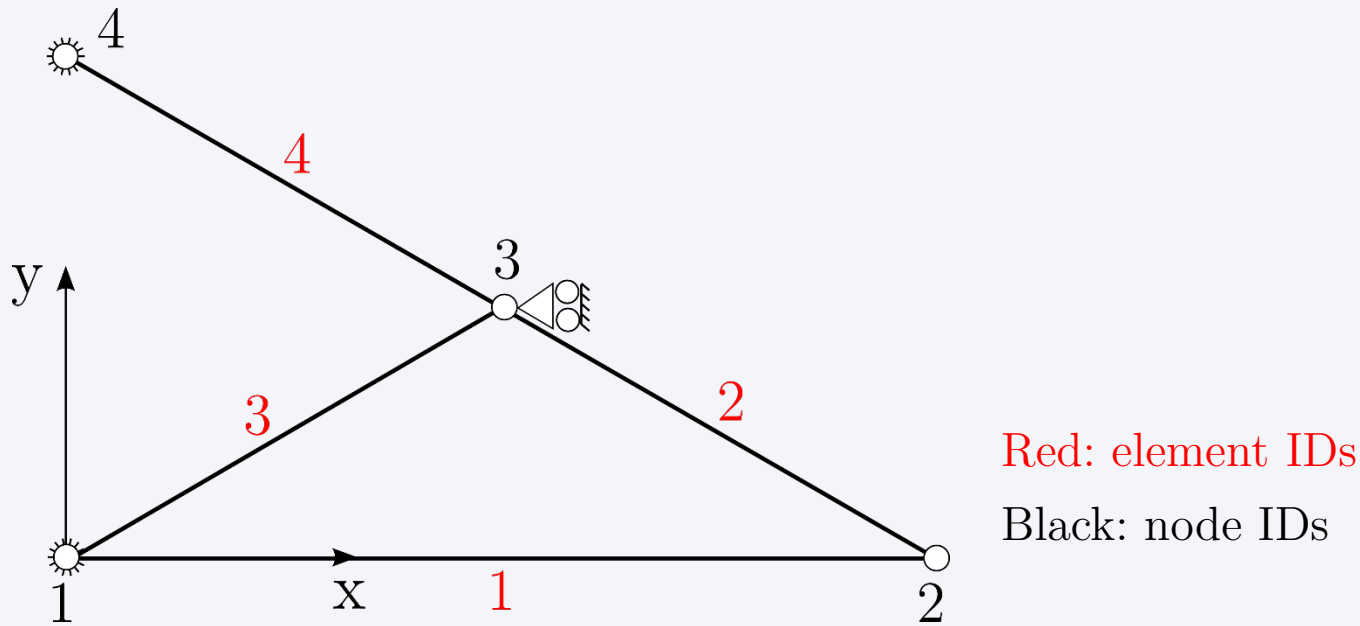
$$P = 10 \text{ kN}$$

$$E = 72 \text{ GPa}$$

$$A = 160 \text{ mm}^2$$

# Example

- The global reference system, the node and element numbering are taken as reported in the figure (these choices are clearly arbitrary)



- Nodal dofs are sorted as:

$$\mathbf{U}^T = \{U_1 \quad V_1 \quad U_2 \quad V_2 \quad U_3 \quad V_3 \quad U_4 \quad V_4\}$$

# Overview of the program – Main

- The structure of the program is very simple, and consists of three distinct steps: the preparation of data (pre-process), the analysis of the structure (solution) and the recovery of those quantities to be used during the analysis of results (post-processing). The three steps are accomplished by the three functions `input_model`, `analyze_structure` and `plot_deformed_shape`

```
% --- 1. Pre-process
INPUT = input_model;

% --- 2. Solution
[ ELEMENTS, NODES, MODEL ] = analyze_structure( INPUT );

% --- 3. Post-process: recovery of forces, plot deformed shapes, ...
ELEMENTS = force_recovery( MODEL, ELEMENTS );
```

# Overview of the program – Structures

- Three main functions are used throughout the program: MODEL, NODES, ELEMENTS. The fields of the structures are here illustrated

```
MODEL =
```

```
struct with fields:
```

ndof: 8	ndof: tot number of dofs for the unconstr system
nels: 4	nels: total number of elements
nnodes: 4	nnodes: total number of nodes
K: [3×3 double]	K: stiffness matrix of constr structure
F: [3×1 double]	F: load vector of constr structure
constr_dofs: [1 2 7 8 5]	constr_dofs: position of constrained dofs
free_dofs: [3 4 6]	free_dofs: position of free dofs
nfree_dofs: 3	nfree_dofs: number of dofs
K_unc: [8×8 double]	K_unc: stiffness matrix of unconstr structure
F_unc: [8×1 double]	F_unc: load vector of unconstr struct
U: [3×1 double]	U: displacement vector (only free dofs)
U_unc: [8×1 double]	U_unc: displacement vector (all dofs)

# Overview of the program – Structures

```
NODES =
```

```
1×4 struct array with fields:
```

coord_x	coord_x: node x coordinate
coord_y	coord_y: node y coordinate
ndof	ndof: number of nodal dofs (2 for 'truss', 6 for 'beams')

# Overview of the program – Structures

```
ELEMENTS =
```

```
1×4 struct array with fields:
```

nodes	nodes: ID of nodes composing the element
EA	EA: axial stiffness
EJ	EJ: bending stiffness
type	type: 'truss' or 'beam'
dofs	dofs: number of element dofs (4: truss; 6: beams)
ptrs	ptrs: vector of pointers
K_el_loc	K_el_loc: element stiffness matrix (local system)
K_el	K_el : element stiffness matrix (global system)
l	l: element length
alpha	alpha: element length
T	T: element transformation matrix
nodal_forces	nodal_forces: element nodal forces

## Step 1, pre-process: input\_model

- The input file is written from the user by specifying the characteristics of the model to be analyzed. The data are organized into the structure INPUT, which is divided into different fields

```
function INPUT = input_model
```

```
INPUT =
```

```
    struct with fields:
```

```
        elements: [4×3 double]
         nodes: [4×3 double]
section_prop: [115200000 0]
        load: [2 2 -10000]
         spc: [5×2 double]
```

```
        elements: [ID_nodeA, ID_nodeB, ID_prop]
         nodes: [ID x y]
section_prop: [EA EJ]
        load: [ID_node direction magnitude]
         spc: [ID_node dof]
```

# Step 1, pre-process: input\_model

- Example of the input file

```
function INPUT = input_model
```

```
% --- Input
```

```
% INPUT.elements      : [ node_A node_B ID_prop ]
% INPUT.nodes         : [ ID_node x_coord y_coord ]
% INPUT.E             : Young's modulus
% INPUT.section_prop  : [ A J ]
%                     : set J = 0 for truss
% INPUT.mass          : [ ID_node component magn]
% INPUT.load          : [ ID_node component magn ]
% INPUT.solution      : 'static' or 'eigenmodes'
% INPUT.spc           : [ ID_node component ]
```

```
% -- Init
```

```
INPUT = struct();
```

```
% -- Elements
```

```
INPUT.elements = [1 2 1;
                  3 2 1;
                  1 3 1;
                  4 3 1];
```

```
% -- Nodes
```

```
INPUT.nodes = [ 1 0 0;
                2 1732 0;
                3 866 500;
                4 0 1000];
```

```
% -- Section properties
```

```
E = 72000;
A1 = 160;
```

```
INPUT.section_prop = [ E*A1 0 ];
```

```
% -- Concentrated mass
```

```
INPUT.mass = [];
```

```
% -- Loading conditions
```

```
INPUT.load = [ 2 2 -10000 ];
```

```
% -- Boundary conditions
```

```
INPUT.spc = [ 1 1
              1 2
              4 1
              4 2
              3 1 ];
```

# Step 1, pre-process: input\_model

- With reference to the present example:

INPUT.elements

1	2	1
3	2	1
1	3	1
4	3	1

Element 1: node 1, node 2, property 1

Element 2: node 3, node 2, property 1

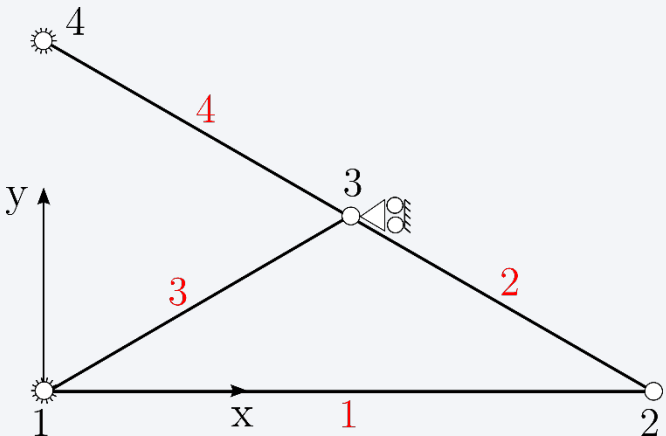
INPUT.nodes

1	0	0
2	1732.0	0
3	866.0	500.0
4	0	1000.0

Node 1: x\_coord, y\_coord

INPUT.spc

1	1	% (node 1, constrained x)
1	2	% (node 1, constrained y)
4	1	% (node 4, constrained x)
4	2	% (node 4, constrained y)
3	1	% (node 3, constrained x)



node 4, constrained y

node 3, constrained x

## Step 2, solution: analyze\_structure

- The function `analyze_structure` represents the core of the program, and is divided into a number of functions

```
function [ ELEMENTS, NODES, MODEL ] = analyze_structure( INPUT )

% --- Set model
[ ELEMENTS, NODES, MODEL ] = set_model( INPUT );

% --- Set pointers
ELEMENTS = set_pointers( ELEMENTS, NODES, MODEL.nels );

% --- Build element stiffness matrices
ELEMENTS = element_stiffness( ELEMENTS, NODES, MODEL.nels );

% --- Assembly stiffness matrix
MODEL = assembly_stiffness( ELEMENTS, MODEL );

% --- Impose constraints and solve
MODEL = solve_structure( MODEL );
```

## Step 2, solution: analyze\_structure

- The function `analyze_structure` represents the core of the program, and is divided into a number of functions

```
function [ ELEMENTS, NODES, MODEL ] = analyze_structure( INPUT )

% --- Set model
[ ELEMENTS, NODES, MODEL ] = set_model( INPUT );

% --- Set pointers
ELEMENTS = set_pointers( ELEMENTS, NODES, MODEL.nels );

% --- Build element stiffness matrices
ELEMENTS = element_stiffness( ELEMENTS, NODES, MODEL.nels );

% --- Assembly stiffness matrix
MODEL = assembly_stiffness( ELEMENTS, MODEL );

% --- Impose constraints and solve
MODEL = solve_structure( MODEL );
```

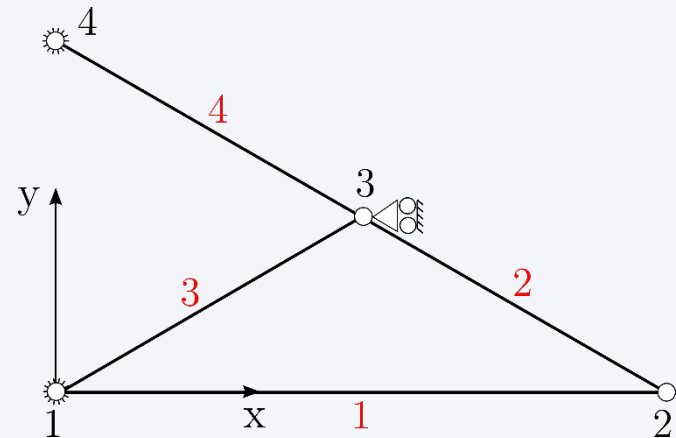
## Step 2, solution: set\_model

- Organize data in a form which is suitable from the solution of the problem. INPUT data are transferred into the structures ELEMENTS, NODES, MODEL

```
function [ ELEMENTS, NODES, MODEL ] = set_model( INPUT )
```

- For example, the fields for element 3 and node 3 are

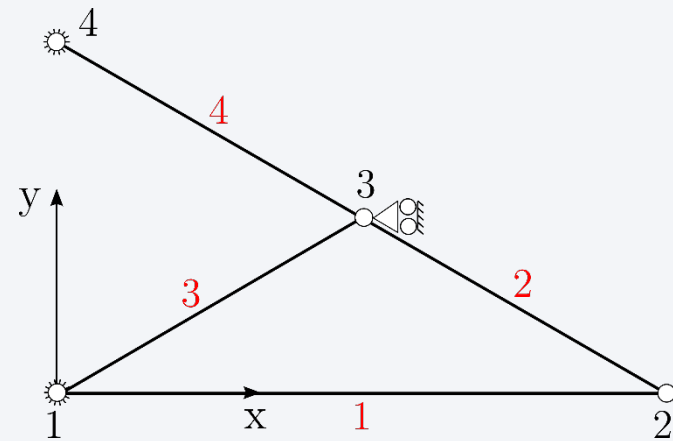
```
>> ELEMENTS(3)
ans =
    struct with fields:
        nodes: [1 3]
        EA: 115200000
        EJ: 0
        type: 'truss'
        dofs: 2
        ptrs: (filled later)
        K_el_loc: (filled later)
        K_el: (filled later)
        l: (filled later)
        alpha: (filled later)
        T: (filled later)
        nodal_forces: (filled later)
```



```
>> NODES(3)
ans =
    struct with fields:
        coord_x: 866.0254
        coord_y: 500
        ndof: 2
```

## Step 2, solution: set\_model

```
>> MODEL
ans =
struct with fields:
    ndof: 8
    nels: 4
    nnodes: 4
    K: [8x8 double]
    F: [8x1 double]
    constr_dofs: [1 2 7 8 5]
    free_dofs: [3 4 6]
    nfree_dofs: 3
    K_unc: (filled later)
    F_unc: (filled later)
    U: (filled later)
    U_unc: (filled later)
```



$$\mathbf{U}^T = \{U_1 \quad V_1 \quad U_2 \quad V_2 \quad U_3 \quad V_3 \quad U_4 \quad V_4\}$$

## Step 2, solution: set\_pointers

```
function [ ELEMENTS, NODES, MODEL ] = analyze_structure( INPUT )

% --- Set model
[ ELEMENTS, NODES, MODEL ] = set_model( INPUT );

% --- Set pointers
ELEMENTS = set_pointers( ELEMENTS, NODES, MODEL.nels );

% --- Build element stiffness matrices
ELEMENTS = element_stiffness( ELEMENTS, NODES, MODEL.nels );

% --- Assembly stiffness matrix
MODEL = assembly_stiffness( ELEMENTS, MODEL );

% --- Impose constraints and solve
MODEL = solve_structure( MODEL );
```

## Step 2, solution: set\_pointers

- The function writes the pointers associated with the generic i-th element in the field `ELEMENT(i).ptrs`

```
ELEMENTS = set_pointers( ELEMENTS, NODES, MODEL.nels )
```

```
>> ELEMENTS(1).ptrs
```

```
ans =
```

```
1      2      3      4
```

```
>> ELEMENTS(2).ptrs
```

```
ans =
```

```
5      6      3      4
```

```
>> ELEMENTS(3).ptrs
```

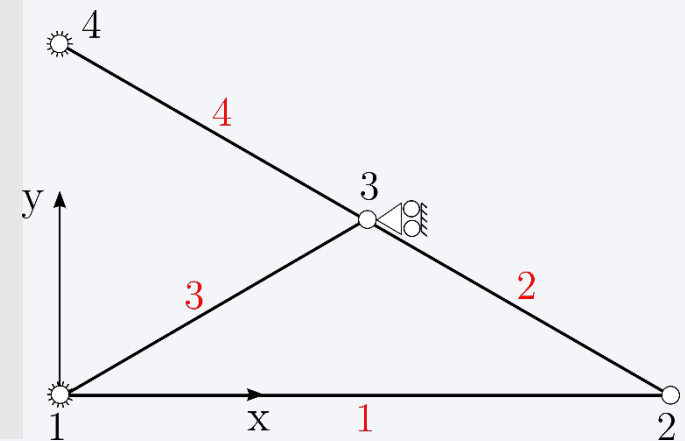
```
ans =
```

```
1      2      5      6
```

```
>> ELEMENTS(4).ptrs
```

```
ans =
```

```
7      8      5      6
```



## Step 2, solution: element\_stiffness

```
function [ ELEMENTS, NODES, MODEL ] = analyze_structure( INPUT )

% --- Set model
[ ELEMENTS, NODES, MODEL ] = set_model( INPUT );

% --- Set pointers
ELEMENTS = set_pointers( ELEMENTS, NODES, MODEL.nels );

% --- Build element stiffness matrices
ELEMENTS = element_stiffness( ELEMENTS, NODES, MODEL.nels );

% --- Assembly stiffness matrix
MODEL = assembly_stiffness( ELEMENTS, MODEL );

% --- Impose constraints and solve
MODEL = solve_structure( MODEL );
```

## Step 2, solution: element\_stiffness (1/3)

```
function ELEMENTS = element_stiffness( ELEMENTS, NODES, n_els )
```

```
for i = 1 : n_els
```

→ Loop over the elements

```
    el_nodes = ELEMENTS(i).nodes;
```

```
    % Determine element length
```

```
    lx = NODES(el_nodes(2)).coord_x - NODES(el_nodes(1)).coord_x;
```

```
    ly = NODES(el_nodes(2)).coord_y - NODES(el_nodes(1)).coord_y;
```

```
    l = sqrt( lx^2 + ly^2 );
```

→ Element length

```
    c = lx / l; % cos( alpha )
```

```
    s = ly / l; % sin( alpha )
```

```
    % Build local stiffness matrix
```

```
    if strcmp( ELEMENTS(i).type, 'truss')
```

```
        % Transformation matrix
```

```
        T = [c s 0 0; 0 0 c s];
```

```
        % Properties and stiffness matrix
```

```
        EA = ELEMENTS(i).EA;
```

```
        ELEMENTS(i).K_el_loc = EA/l*[1 -1; -1 1];
```

→ Stiffness matrix in local coord

## Step 2, solution: element\_stiffness (2/3)

```
elseif strcmp( ELEMENTS(i).type, 'beam')

    % Transformation matrix
    T_node = [c s 0; -s c 0; 0 0 1];
    T = [T_node zeros(3,3); zeros(3,3) T_node];

    % Properties and stiffness matrix
    EA = ELEMENTS(i).EA;
    EJ = ELEMENTS(i).EJ;

    K_aa = [ EA/l 0 0;
            0 12*EJ/l^3 6*EJ/l^2;
            0 6*EJ/l^2 4*EJ/l];

    K_ab = [-EA/l 0 0;
            0 -12*EJ/l^3 6*EJ/l^2;
            0 -6*EJ/l^2 2*EJ/l];

    K_bb = [EA/l 0 0;
            0 12*EJ/l^3 -6*EJ/l^2;
            0 -6*EJ/l^2 4*EJ/l];

    ELEMENTS(i).K_el_loc = [K_aa K_ab; K_ab' K_bb]; → Stiffness matrix in local coord

end
```

## Step 2, solution: element\_stiffness (3/3)

```
% Rotate stiffness matrix
```

```
ELEMENTS(i).K_el = T' * ELEMENTS(i).K_el_loc * T; → Stiffness matrix in global coord
```

```
% Store some useful values
```

```
ELEMENTS(i).l = l;
```

```
ELEMENTS(i).alpha = atan2(s,c)*180/pi;
```

```
ELEMENTS(i).T = T;
```

```
end
```

## Step 2, solution: assembly\_stiffness

```
function [ ELEMENTS, NODES, MODEL ] = analyze_structure( INPUT )

% --- Set model
[ ELEMENTS, NODES, MODEL ] = set_model( INPUT );

% --- Set pointers
ELEMENTS = set_pointers( ELEMENTS, NODES, MODEL.nels );

% --- Build element stiffness matrices
ELEMENTS = element_stiffness( ELEMENTS, NODES, MODEL.nels );

% --- Assembly stiffness matrix
MODEL = assembly_stiffness( ELEMENTS, MODEL );

% --- Impose constraints and solve
MODEL = solve_structure( MODEL );
```

## Step 2, solution: assembly\_stiffness

- The assembly of the stiffness matrix is readily performed by using the vectors of pointers to directly set the contribution of each element in the correct position of the global stiffness matrix

```
function MODEL = assembly_stiffness( ELEMENTS, MODEL )

% --- Assembly stiffness matrix
for i = 1 : MODEL.nels

    ptrs = ELEMENTS( i ).ptrs;
    K_el = ELEMENTS( i ).K_el;

    MODEL.K( ptrs, ptrs ) = MODEL.K( ptrs, ptrs ) + K_el;

end
```

## Step 2, solution: solve\_structure

```
function [ ELEMENTS, NODES, MODEL ] = analyze_structure( INPUT )

% --- Set model
[ ELEMENTS, NODES, MODEL ] = set_model( INPUT );

% --- Set pointers
ELEMENTS = set_pointers( ELEMENTS, NODES, MODEL.nels );

% --- Build element stiffness matrices
ELEMENTS = element_stiffness( ELEMENTS, NODES, MODEL.nels );

% --- Assembly stiffness matrix
MODEL = assembly_stiffness( ELEMENTS, MODEL );

% --- Impose constraints and solve
MODEL = solve_structure( MODEL );
```

## Step 2, solution: solve\_structure

```
function MODEL = solve_structure( MODEL )

constr_dofs = MODEL.constr_dofs;

% Store unconstrained K and F
MODEL.K_unc = MODEL.K;
MODEL.F_unc = MODEL.F;

% Impose constraints
MODEL.K( constr_dofs, : ) = []; —————> Remove rows of constrained dofs
MODEL.K( :, constr_dofs ) = []; —————> Remove cols of constrained dofs
MODEL.F( constr_dofs ) = []; —————> Remove rows of constrained dofs

% Solve problem
MODEL.U = MODEL.K \ MODEL.F;

% Expand displacements to the global vector
MODEL.U_unc = zeros( MODEL.ndof, 1);
MODEL.U_unc( MODEL.free_dofs ) = MODEL.U;
```

## Step 2, solution: solve\_structure

- Here below the stiffness matrix, the load vector and the displacement for comparison purposes

```
>> MODEL.K
```

```
ans =
```

```
1.0e+05 *
```

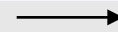
```
1.5291    -0.4988    0.4988  
-0.4988    0.2880   -0.2880  
0.4988   -0.2880    0.8640
```

```
>> MODEL.F
```

```
ans =
```

```
0  
-10000  
0
```

```
>> MODEL.U
```



displacements of  
unconstrained dofs

```
ans =
```

```
-0.2604  
-0.9719  
-0.1736
```

## Step 3, post-process: force\_recovery

```
function ELEMENTS = force_recovery( MODEL, ELEMENTS )
```

```
% --- Force recovery
```

```
for i = 1 : MODEL.nels
```

```
    T = ELEMENTS(i).T;
```

```
    ptrs = ELEMENTS(i).ptrs;
```

```
    U_el_loc = T * MODEL.U_unc( ptrs );
```

→ Displacements of the element in local coordinates

```
    nodal_forces = ELEMENTS(i).K_el_loc * U_el_loc;
```

```
    %(take force in node 2: >0 in traction)
```

```
    ELEMENTS(i).nodal_forces = nodal_forces(2);
```

```
end
```

## Step 3, post-process: force\_recovery

```
ELEMENTS.nodal_forces
```

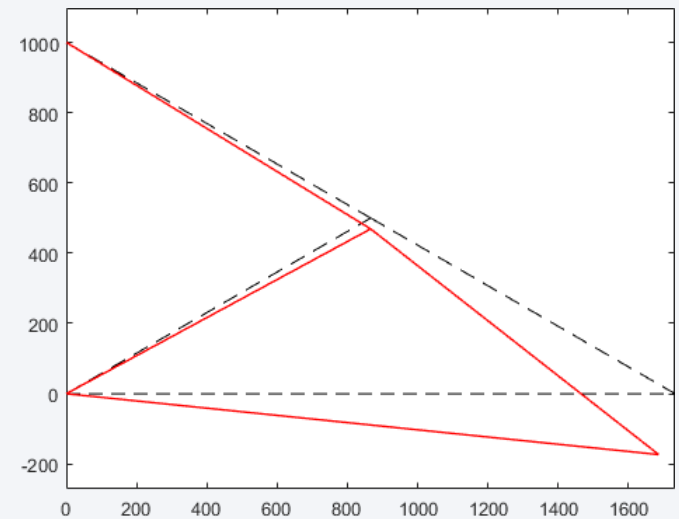
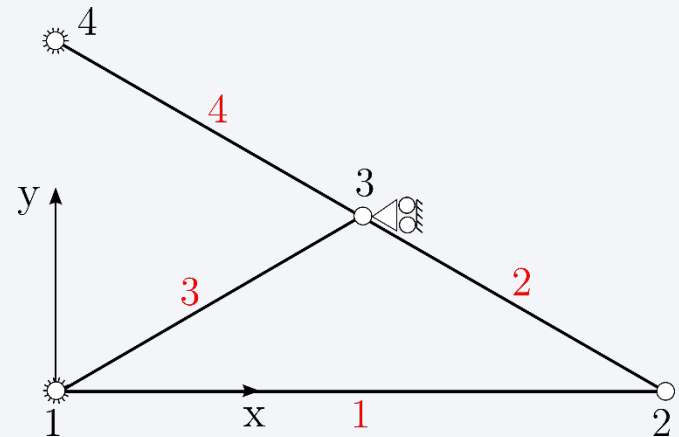
```
ans =
```

```
-1.7321e+04 → axial force in element 1  
2.0000e+04   (<0: compression)  
-1.0000e+04  
1.0000e+04 → axial force in element 4  
(>0: traction)
```

```
MODEL.U_unc
```

```
ans =
```

```
0  
0  
-0.2604  
-0.9719  
0  
-0.1736  
0  
0
```

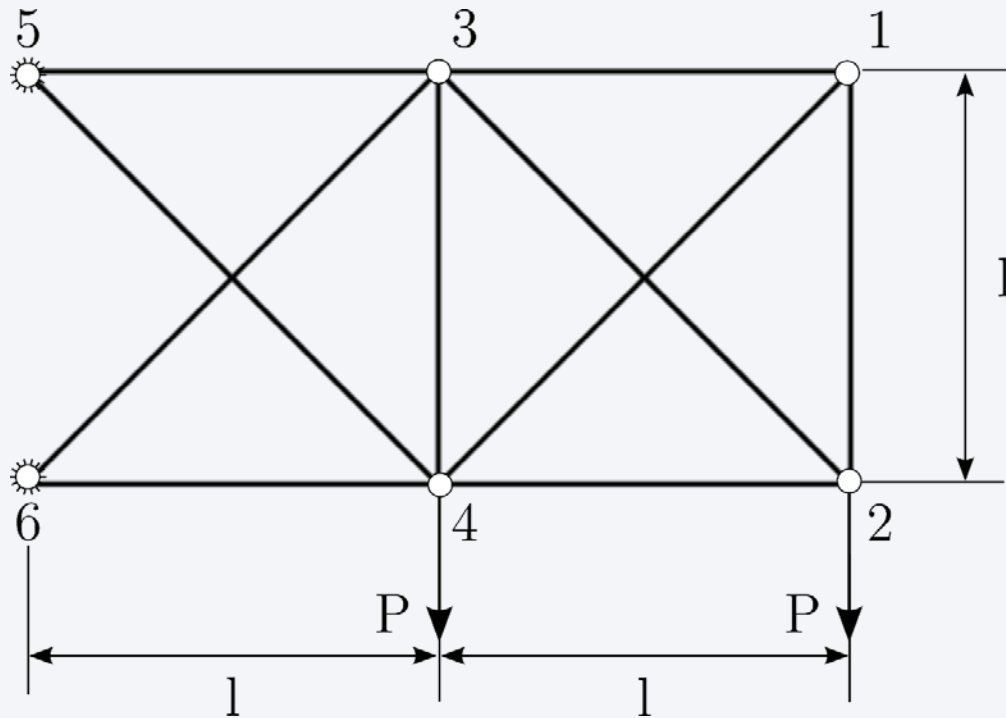


# Exercises

- Complete the program by writing the missing parts of the code
- Extend to program to include the possibility of performing free-vibration analyses
- Solve the problems reported in the next slides and check the correctness of the implementation

## Exercise 1 – 10 truss structure

- Evaluate the deformed shape and the free vibrations of the structure in the figure



### Input data

$$l = 360 \text{ mm}$$

$$P = 100 \text{ N}$$

$$E = 1e4 \text{ MPa}$$

$m = 1e-4 \text{ t}$  (lumped mass at each node)

Square section of dimension

$$a = \sqrt{10} \text{ mm}$$

# Exercise 1 – deformed configuration

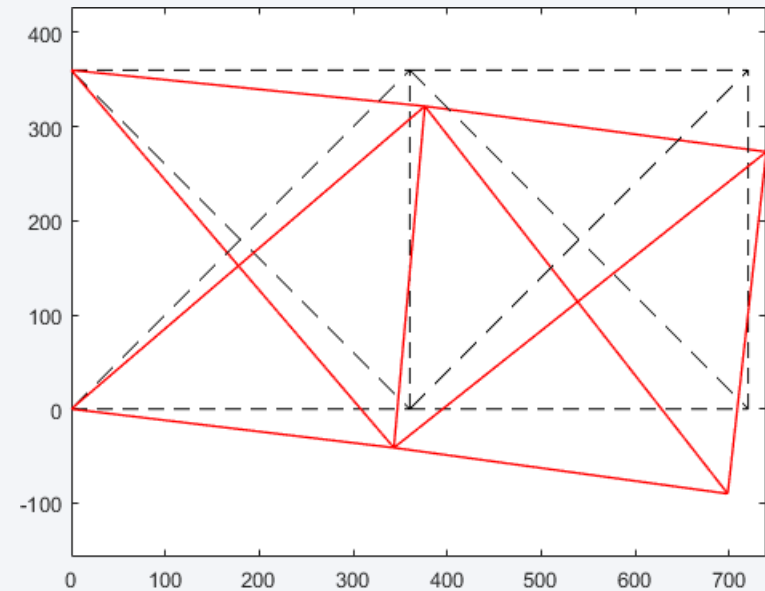
- Solution of the linear static problem

$$\mathbf{KU} = \mathbf{f}$$

```
>> MODEL.U
```

```
ans =
```

```
0.8478  
-3.7951  
-0.9522  
-3.9396  
0.7033  
-1.6744  
-0.7367  
-1.8021
```



Deformed shape

# Exercise 1 – eigenvalues and eigenvectors

- Solution of the eigenvalue problem

$$(-\omega^2 \mathbf{M} + \mathbf{K}) = \mathbf{0}$$

```
>> sqrt(diag(MODEL.Ome2))
```

```
ans =
```

```
1.0e+03 *
```

```
0.3764
```

```
1.1468
```

```
1.2102
```

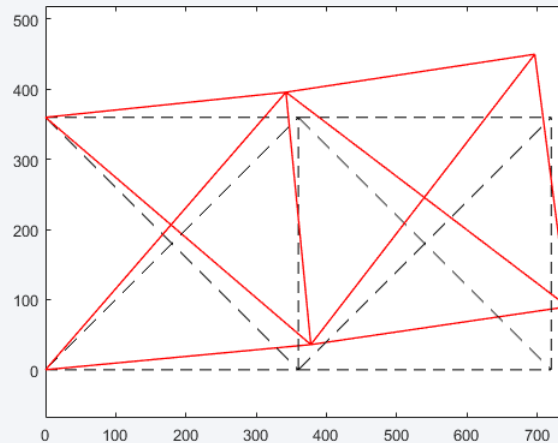
```
2.0797
```

```
2.3908
```

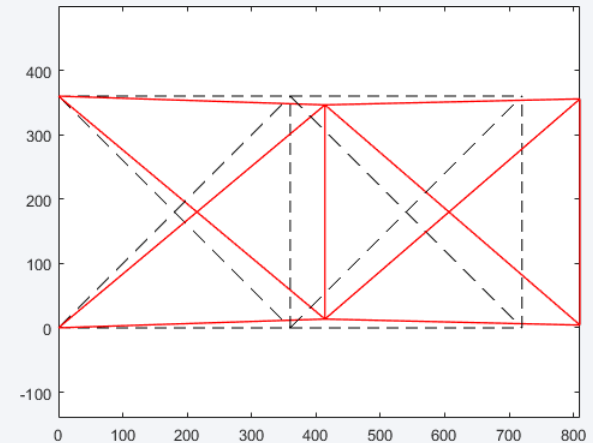
```
2.7817
```

```
2.8800
```

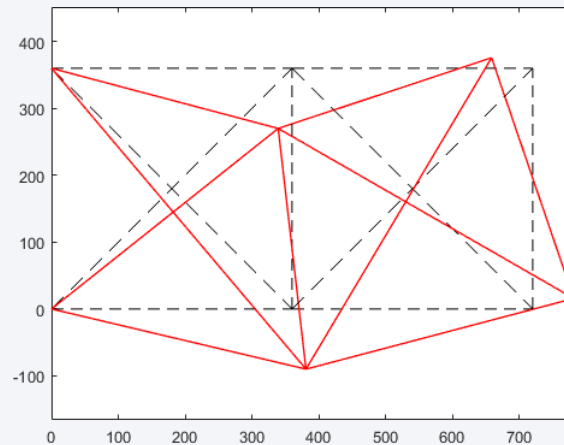
```
3.2508
```



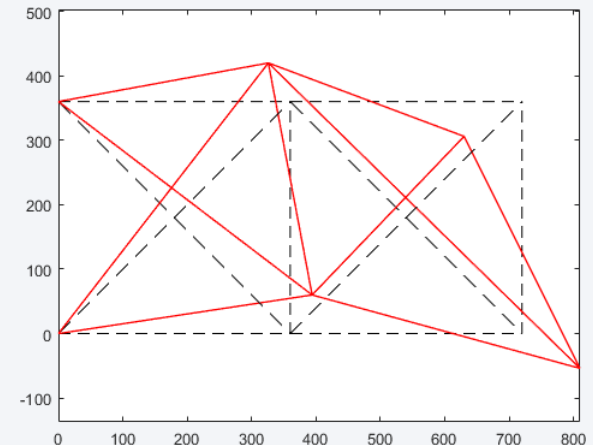
Mode 1



Mode 2



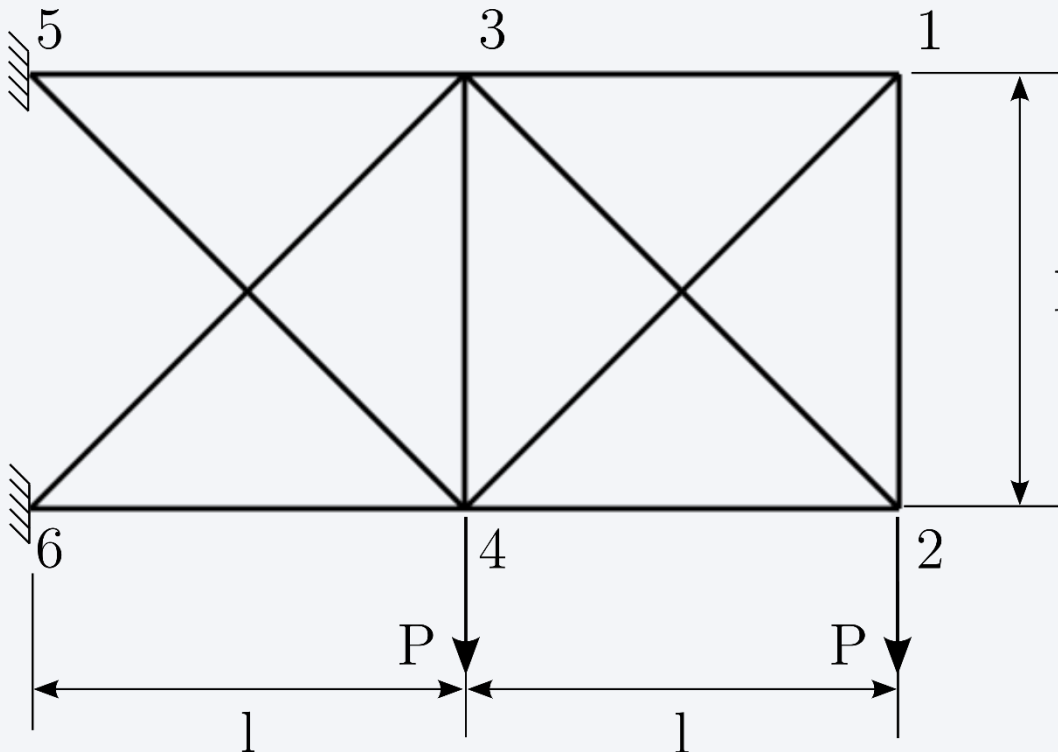
Mode 3



Mode 4

## Exercise 1 – beam model structure

- Consider now a model of the structure by using beam instead of truss elements



- Each node is thus characterized by displacement and rotation dofs
- The boundary condition implies that the both displacements and rotations are set to zero
- The presence of an axial load path suggests that no differences are expected with respect to the truss model

# Exercise 1 – beam model structure

- Comparison of results

Using beam elements

```
>> MODEL.U  
  
ans =  
  
    0.8477 (Ux1)  
   -3.7948 (Uy1)  
   -0.0053 (th1)  
   -0.9522 (Ux2)  
   -3.9392 (Uy2)  
   -0.0055 (th2)  
    0.7033 (Ux3)  
   -1.6741 (Uy3)  
   -0.0054 (th3)  
   -0.7366 (Ux4)  
   -1.8019 (Uy4)  
   -0.0055 (th4)
```

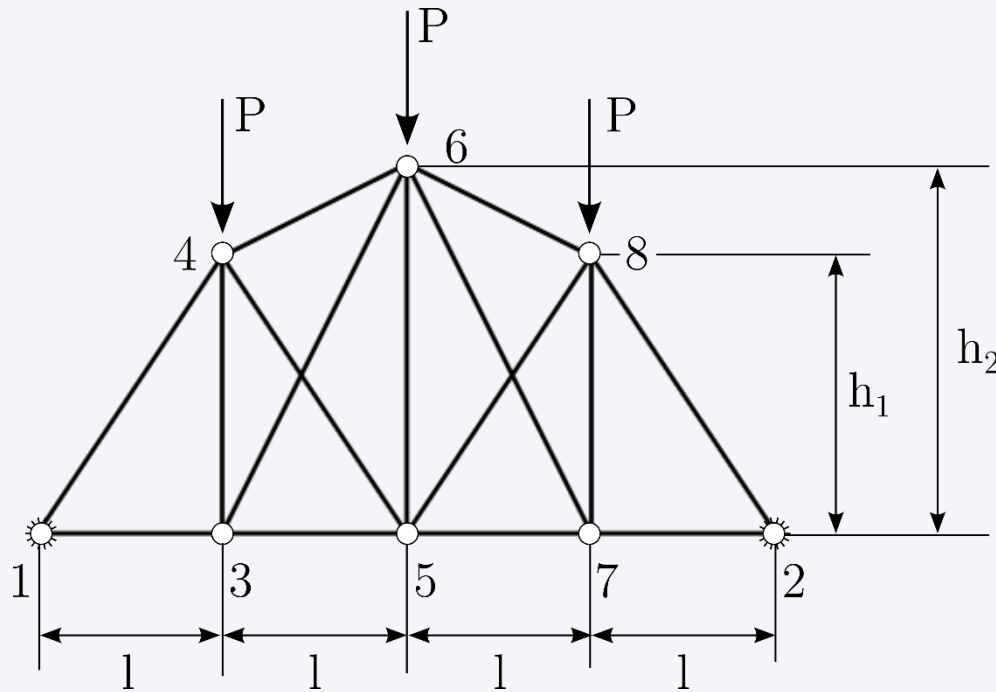
Using truss elements

```
>> MODEL.U  
  
ans =  
  
    0.8478 (Ux1)  
   -3.7951 (Uy1)  
   -0.9522 (Ux2)  
   -3.9396 (Uy2)  
    0.7033 (Ux3)  
   -1.6744 (Uy3)  
   -0.7367 (Ux4)  
   -1.8021 (Uy4)
```

- Note that the nodal displacements are very similar. Neglecting the bending stiffness does not affect the quality of the predictions as the response is axially-dominated

## Exercise 2 – 15 truss structure

- Evaluate the deformed shape and the free vibrations of the structure in the figure



### Input data

$$l = 2540 \text{ mm}$$

$$h_1 = 3810 \text{ mm}$$

$$h_2 = 5080 \text{ mm}$$

$$P = 35 \text{ N}$$

$$E = 200 \text{ GPa}$$

$$A = 10 \text{ mm}^2$$

$$m = 1\text{e-}4 \text{ t (lumped mass at each node)}$$

## Exercise 2 – deformed configuration

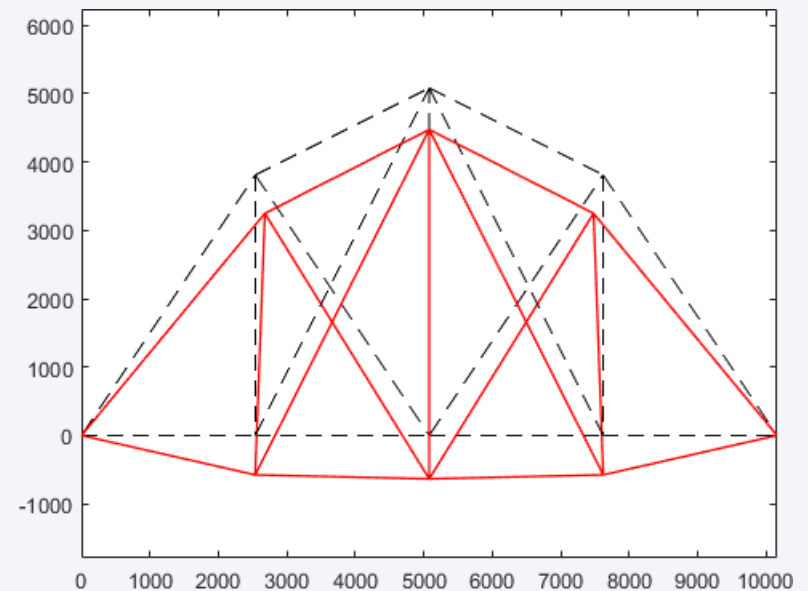
- Solution of the linear static problem

$$\mathbf{KU} = \mathbf{f}$$

```
>> MODEL.U
```

```
ans =
```

```
-0.0009  
-0.2134  
0.0515  
-0.2079  
0.0000  
-0.2351  
0.0000  
-0.2241  
0.0009  
-0.2134  
-0.0515  
-0.2079
```



Deformed shape

## Exercise 2 – eigenvalues and eigenvectors

- Solution of the eigenvalue problem

$$(-\omega^2 \mathbf{M} + \mathbf{K}) = \mathbf{0}$$

```
>> sqrt(diag(MODEL.Ome2))
```

```
ans =
```

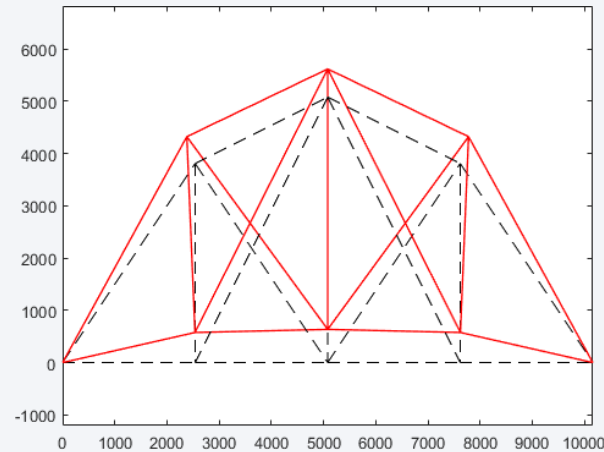
```
1.0e+03 *
```

```
0.8574
```

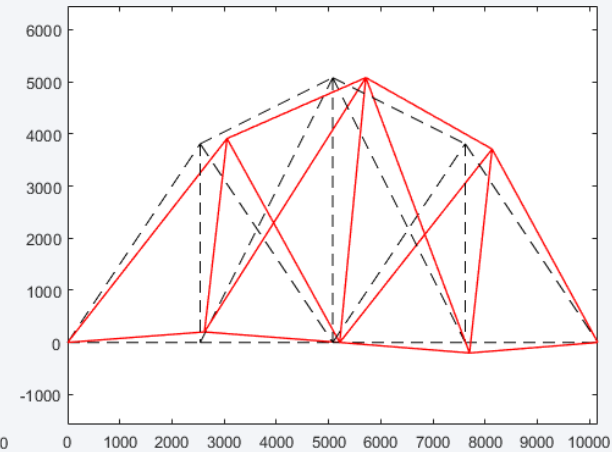
```
1.2536
```

```
1.5817
```

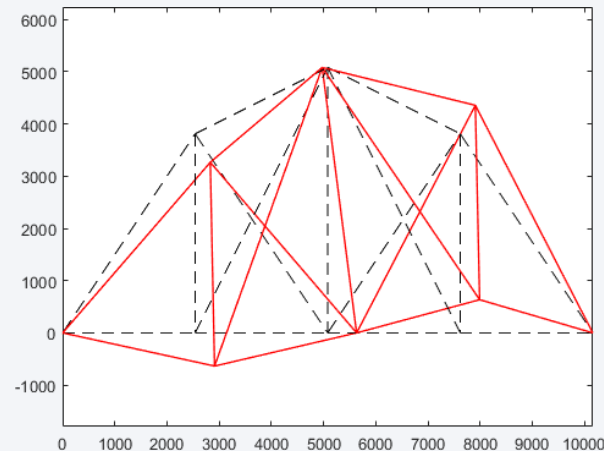
```
2.2137
```



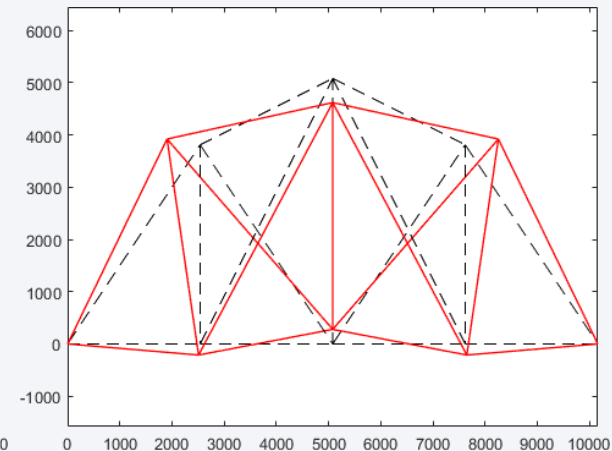
Mode 1



Mode 2



Mode 3



Mode 4