AME50541: Finite Element Methods
Homework 1: Due Friday, February 1 2019

Problem 1: (10 points) JNR 4.1 (uploaded to Sakai if needed)

Problem 2: (20 points) Solve for the displacement of the nodes of the truss 0 (Figure 1) using equilibrium at each node and Hooke's law. The Young's modulus times the cross-sectional area of each element are: $EA_e = e$ for $e = 1, \ldots, 5$. Setup the linear system of equations by hand and use MATLAB to solve it. Report the displacements and forces of all nodes.

Problem 3: (20 points) Solve for the displacement of the nodes of the truss 1 (Figure 1) using equilibrium at each node and Hooke's law. The spring is at rest when the truss is in its undeformed configuration (Figure 1). The Young's modulus times the cross-sectional area of each element are: $EA_e = e$ for $e = 1, \ldots, 5$ and the stiffness of the spring is $k = 1$. Recall the force in a spring is $F = k\Delta x$ where Δx is the deformation of the spring from its rest configuration. Setup the linear system of equations by hand and use MATLAB to solve it. Report the displacements and forces of all nodes.

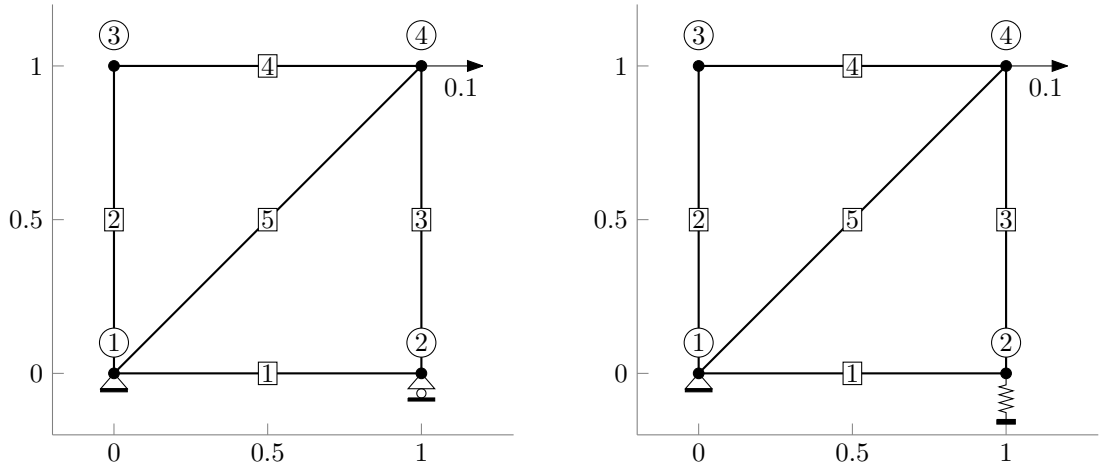

Figure 1: Truss 0 (*left*) and Truss 1 (*right*)

Problem 4: (50 points) In this problem, you will implement the direct stiffness method in a series of steps. Before proceeding, carefully review the below code/comments as they provide crucial information regarding the specification of the truss topology, material properties, and boundary conditions, which will be needed to implement the direct stiffness method. For concreteness, the code below defines truss 0 (Figure 1).

```
function [xcg, e2vcg, EA, dbc_idx, dbc_val, fbc_val] = define_truss0(plotit)

if nargin<1, plotit = false; end

%Definitions
%------------
%ndim : Number of spatial dimensions (planar truss: ndim = 2)
%nnode : Number of nodes in the truss
%nelem : Number of elements in the truss
%nnode_per_elem : Number of nodes per element (truss: nnode_per_elem = 2)
%ndbc : Number of degrees of freedom with Dirichlet boundary conditions
%ndof_per_node: Number of degrees of freedom per node
%               (truss: ndof_per_node = ndim)
%ndof : Number of global degrees of freedom (ndof = ndof_per_node*nnode)

%XCG : 2D array (ndim, nnode) : The position of the nodes in the truss
```

```
%prior to deformation under the external loads. The (i, j)—entry is the
%position of node j in the ith dimension. The global node numbers are
%defined by the columns of this matrix, e.g., the node at
%xcg(:, j) is the jth node of the truss.
xcg = [0.0, 1.0, 0.0, 1.0; ...
       0.0, 0.0, 1.0, 1.0];

%E2VCG : 2D array (nnode_per_elem, nelem): The connectivity of the
%truss. The (:, e)—entries are the global node numbers of the nodes that
%comprise element e. The local node numbers of each element are defined
%by the columns of this matrix, e.g., e2vcg(i, e) is the global node
%number of the ith local node of element e.
e2vcg = [1, 1, 2, 3, 1; ...
         2, 3, 4, 4, 4];

%EA : Array (nelem,) : Young's modulus times cross—sectional area for
%each element.
EA = [1.0; 2.0; 3.0; 4.0; 5.0];

%DBC_IDX : Array (ndbc,) : Indices into array defined over global dofs
%(size = ndim*nnode) that indicates those with prescribed displacements
%(Dirichlet BCs).
dbc_idx = [1; 2; 4];

%DBC_VAL : Array (ndbc,) : Value of the prescribed displacements such
%that U(DBC_IDX) = DBC_VAL, where U is an array of size ndim*nnode that
%contains the displacement of each node in the truss with components
%U = [Ux_1; Uy_1; ... ; Ux_nnode; Uy_nnode], where Ux_i, Uy_i are the x—
% and y— displacements at node i.
dbc_val = [0.0; 0.0; 0.0];

%FBC_VAL : Array (nfbc,) : Value of the prescribed forces at all global
%dofs without a prescribed displacement (nfbc = ndim*nnode—ndbc). Let
%FBC_IDX = setdiff(1:NDIM*NNODE, DBC_IDX), then F(FBC_IDX) = FBC_VAL,
%where F is an array of size ndim*nnode that contains the force at each
%node in the truss with components F = [Fx_1; Fy_1; ... ; Fx_nnode; Fy_nnode],
%where Fx_i, Fy_i are the x— and y— forces at node i.
fbc_val = [0.0; 0.0; 0.0; 0.1; 0.0];

% Plot truss
if plotit
    [ndim, nnode] = size(xcg);
    f = zeros(ndim*nnode, 1);
    fbc_idx = setdiff(1:ndim*nnode, dbc_idx);
    f(fbc_idx) = fbc_val;
    visualize_truss(zeros(ndim*nnode,1), xcg, e2vcg, EA, 1, f, dbc_idx);
end

end
```

Problem 4.1 Implement a function that creates two MATLAB structures that define the truss element and its data. It should have the following signature

```
function [elem, elem_data] = create_elem_structs_truss(xcg, e2vcg, EA)
%CRETE_ELEM_STRUCTS_TRUSS Create structures defining the truss element and its
%data.
%
%Input arguments
%———————————————
%   XCG, E2VCG, EA : See description in SOLVE_TRUSS_DSM
%
%Output arguments
%————————————————
%   ELEM : structure with fields NDOF_PER_NODE, NNODE_PER_ELEM, STIFF
%     ELEM.NDOF_PER_NODE : number : Number of degrees of freedom per node
```

```
%     ELEM.NNODE_PER_ELEM : number : Number of nodes per element
%     ELEM.STIFF : function : Function that takes a single entry of the
%     ELEM_DATA structure array as input and returns the element stiffness
%     matrix.
%
%   ELEM_DATA : structure array (nelem,) with element-specific fields
%   (for truss: EA_OVER_L, STH, CTH)
%      ELEM_DATA(e).EA_OVER_L : number : EA/L for element e
%      ELEM_DATA(e).STH : number : sine of the angle element e makes with
%      the horizontal
%      ELEM_DATA(e).CTH : number : cosine of the angle element e makes with
%      the horizontal
```

Starter code is provided on the course website in the Homework 1 code distribution: `create_elem_structs_truss.m`. Use truss 0 and your work from Problem 2 to test your function. Report $\sin\theta$, $\cos\theta$, and EA/L for each element of truss 0.

Problem 4.2 Implement a function that evaluates the stiffness matrix for a truss element. It should have the following signature

```
function [Ke] = eval_elem_contrib_truss(elem_data)
%EVAL_ELEM_CONTRIB_TRUSS Evaluate the stiffness matrix for a truss element.
%
%Input arguments
%----------------
%   ELEM_DATA : Single entry ELEM_DATA described in CREATE_ELEM_STRUCTS_TRUSS
%
%Output arguments
%-----------------
%   KE : 2D matrix (ndof_per_node*nnode_per_elem,
%                   ndof_per_node*nnode_per_elem)
%      : Truss element stiffness matrix
```

Starter code is provided on the course website in the Homework 1 code distribution: `eval_elem_contrib_truss.m`. Use truss 0 and your work from Problem 2 to test your function. Report the stiffness matrix for element 2 of truss 0.

Problem 4.3 Implement a function that evaluates and stores the element stiffness matrix for each member in the truss. It should have the following signature

```
function  [Ke] = eval_stiff_unassembled(elem, elem_data)
%EVAL_STIFF_UNASSEMBLED Evaluate/store element stiffness matrix for each
%element.
%
%Input arguments
%----------------
%   ELEM, ELEM_DATA : See description in CREATE_ELEM_STRUCTS_TRUSS
%
%Output arguments
%-----------------
%   Ke : 3D array (ndim*nnode_per_elem, ndim*nnode_per_elem, nelem) :
%   unassembled element stiffness matrices (Ke(:, :, e) is the stiffness
%   matrix of element e).
```

This function signature and these comments are provided in a file on the course website in the Homework 1 code distribution: `eval_stiff_unassembled.m`.

Problem 4.4 Implement a function that creates a matrix that maps local degrees of freedom for each element to global degrees of freedom (ignoring boundary conditions). It should have the following signature

```
function [ldof2gdof] = create_map_ldof_to_gdof(ndof_per_node, e2vcg)
%CREATE_MAP_LDOF_TO_GDOF Create a matrix that maps local degrees of freedom
%for each element to global degrees of freedom (ignoring boundary
%conditions).
%
%Input arguments
%----------------
%   NDOF_PER_NODE : Number of degrees of freedom per node
%
%   E2VCG : See description in SOLVE_TRUSS_DSM
%
%Output arguments
%-----------------
%    LDOF2GDOF : 2D array (ndof_per_node*nnode_per_elem, nelem) : Maps
%    local degrees of freedom for each element to global degrees of
%    freedom, ignoring boundary conditions. LDOF2GDOF(i, e) is the global
%    degree of freedom corresponding to the ith local degree of freedom of
%    element e.
```

Starter code is provided on the course website in the Homework 1 code distribution: `create_map_ldof_to_gdof.m`. Use truss 0 to test your function and report the `ldof2gdof` matrix for truss 0.

Problem 4.5 Implement a function that assembles the element stiffness matrices into the global stiffness matrix without applying Dirichlet boundary conditions. It should have the following signature

```
function [K] = assemble_stiff_nobc(ndof, Ke, ldof2gdof)
%ASSEMBLE_STIFF_NOBC Assemble element stiffness matrices into the global
%stiffness matrix without applying Dirichlet boundary conditions.
%
%Input arguments
%----------------
%   NDOF : Number of global degrees of freedom (= ndof_per_node*nnode)
%   KE : See defintion in EVAL_STIFF_UNASSEMBLED
%   LDOF2GDOF : See definition in CREATE_MAP_LDOF_TO_GDOF
%
%Output arguments
%-----------------
%   K : 2D array (ndim*nnode, ndim*nnode) : assembled stiffness matrix
%   PRIOR to static condensation.
```

This function signature and these comments are provided in a file on the course website in the Homework 1 code distribution: `assemble_stiff_nobc.m`. Use truss 0 and your work from Problem 2 to test your function.

Problem 4.6 Implement a function that applies boundary conditions via static condensation to the global stiffness matrix and solves for the unknown displacements and reaction forces. It should have the following signature

```
function  [u, f] = apply_bc_solve(K, dbc_idx, dbc_val, fbc_val)
%APPLY_BC_SOLVE Apply boundary conditions via static condensation and
%solve for unknown displacements and reaction forces.
%
%Input arguments
%----------------
%   K : 2D array (ndim*nnode, ndim*nnode) : assembled stiffness matrix
%   (output of ASSEMBLE_STIFF_NOBC; see complete description in that
%   function).
%
%   DBC_IDX, DBC_VAL, FBC_VAL : See defintion in SOLVE_TRUSS_DSM
%
%Output arguments
%-----------------
%   U, F : See defintion in SOLVE_TRUSS_DSM
```

This function signature and these comments are provided in a file on the course website in the Homework 1 code distribution: `apply_bc_solve.m`. Use truss 0 and your work from Problem 2 to test your function.

Problem 4.7 Implement a function that uses the direct stiffness method to solve for the nodal displacements and reaction forces of a truss structure using the functions created in Problems 4.1-4.2. It should have the following signature

```
function [u, f] = solve_truss_dsm(xcg, e2vcg, EA, dbc_idx, dbc_val, fbc_val)
%SOLVE_TRUSS_DSM Solve for the nodal displacements and reaction forces of a
%truss structure.
%
% Input arguments
% ———————————————
%   XCG, E2VCG, EA, DBC_IDX, DBC_VAL, FBC_VAL : See definition in
%   DEFINE_TRUSS0
%
% Output arguments
% ————————————————
%   U : Array (ndim*nnode,) : The displacement of each node in the truss
%   with components U = [Ux_1; Uy_1; ... ; Ux_nnode; Uy_nnode], where Ux_i,
%   Uy_i are the x— and y— displacements at node i.
%
%   F : Array (ndim*nnode,) : The force acting on each node of the truss
%   with components F = [Fx_1; Fy_1; ... ; Fx_nnode; Fy_nnode], where Fx_i,
%   Fy_i are the x— and y—forces at node i.
%
```

Starter code is provided on the course website in the Homework 1 code distribution: `solve_truss_dsm.m`. Use truss 0 and your work from Problem 2 to test your function. Report the displacements and forces of all nodes (these should agree with your answer for Problem 2).

Problem 5: (10 points) Use the functions written in Problem 4 to solve for the nodal displacements and reaction forces of truss 2 (Figure 2). Report the displacements and forces at each node and plot the deformed truss using the function `visualize_truss` provided on the course website. This requires implementing a new function to replace `define_truss0` that defines the topology, material properties, and boundary conditions of this truss and then passing the resulting variables to your function `solve_truss_dsm`.

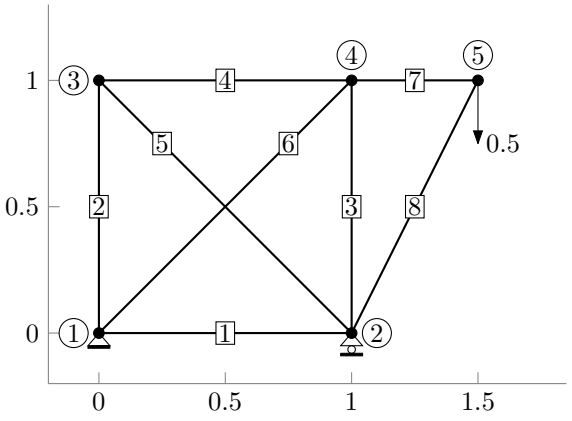

Figure 2: Truss 2

Problem 6: (10 points) Use the functions written in Problem 4 to solve for the nodal displacements and reaction forces of the Warren truss (Figure 3). Report the displacements of the node at the top right of the truss (node with the horizontal external force) and forces on the node at the bottom left of the truss (pinned

node). The nodal coordinates, connectivity, boundary conditions, and load are defined in the function `define_warren_truss` that can be found in the Homework 1 code distribution on the course website.

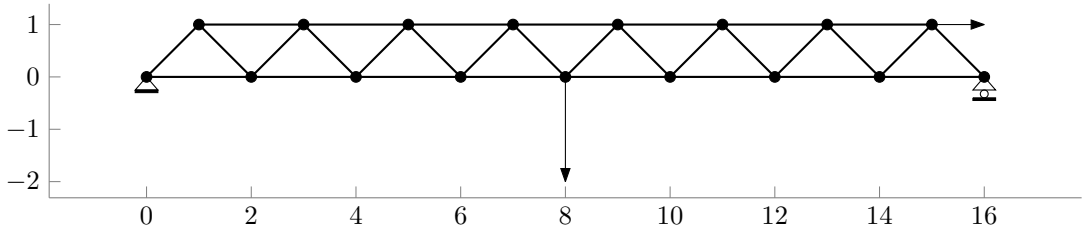

Figure 3: Warren truss