CEE 235A: Adv. Structural Analysis

FEM Project

Elmer Wu & Khalid Alsadhan

ABSTRACT

This report is a final submission of FEM code for 3D frame members accounting for shear deformation. It includes a brief rundown of the code and includes verification problems.

FEM Project Final Submission

Program Features and Overall Structure

We discuss the basic features of the program by splitting up the "main" program (ud_3d1el.m) and all the related sub-routines (listed below). The goal is to develop a 3D FEM code for frames that accounts for shear deformations and integrates with MASTAN2's 1st-order elastic user defined analysis function. Flow charts will follow the brief description of the code.

ud_3d1el.m

This is the main routine that performs the 3D FEM analysis on the MASTAN2 structure. To simplify the flow of the information, bullet points will be used.

- Generate mapping for the degree of freedom
  - Create node_DOF and memb_id for mapping using MASTAN "ends"
- Extract concentrated load vector from "concen" array from MASTAN2
  - Output as appliedNodalLoads
- Convert "fixities" to column vector D
  - Extract "supportDOF", "displacedDOF", and "freeDOF" information by utilizing intrinsic functions such as "find(isnan)", etc.
- Create an array for the fixed end forces of each member
  - Use element ends as coordi and coordj to call lengthfunction.m for L
  - Create member FEFs through subroutine kaewu_computeMemberFEFs.m
  - Use kaewu_etran.m to transform the fixed end forces to global cords
  - Use "memb_id" to assemble the FEF vector to be used later
- Assemble global structure stiffness matrix by looping across all # elements
  - Call lengthfunction.m to pass to stiffness assembly
  - Use kaewu_estiff.m to create local stiffness matrix given the geometric properties of the current element *i*
  - Transform from local stiffness to global stiffness using kaewu_etran.m
  - Use "memb_id" information to assemble "k_structure_global"
- Extract Kff, Ksf, Ksn, etc. from the global stiffness by using the "supportDOF", "displacedDOF", and "freeDOF" indices found when converting fixities
- Use equations learned in class to solve for reactions "R" and displacements "D" for the indices in question by combining stiffness sub-matrix information with loading sub-vectors
- Compute member forces by reintroducing the displacement results back into the member stiffness matrix
- Reshape the results into "REACT" and "DEFL" for use by MASTAN2

lengthfunction.m

The length of an element is a parameter that is passed to multiple sub-routines and used multiple times in various parameters. Thus, we developed a separate standalone function that receives the coordinates of the ends of the members and outputs the length of the member using the distance formula.

Inputs:	coordi(ends(i,1))	coordj(ends(i,2))
Outputs:	L	

kaewu_computeMemberFEFs.m

This function calculates the local coordinate fixed end forces given distributed loads. These fixed end forces are reaction forces to the loading condition and located at the end nodes of a member. Thus, they are opposite in sign and magnitude to the actual local member loading. This information will be used in conjunction with **appliedNodalLoads** array for the [P] vector that is used to find displacements when $\{u\} = [K]^{-1}(\{P\} - \{FEF\})$

Inputs:	w(i)	L
Outputs:	{FeF}	

kaewu_estiff.m

This function calculates the element stiffness matrices in local coordinates per member as the main code loops across all members. The subroutine takes into account shear deformations and incorporates them into the traditional stiffness matrix. Specifically, $F_{y,i}$, $M_{z,i}$, $F_{y,j}$, and $M_{z,j}$ are changed with shear deformation. The first subscript corresponds to the axis direction and the second subscript corresponds to which end of the member.

Inputs:	geometric element properties (A, I_y, I_z, J, A_y, A_z, v, L)
Outputs:	elk(12:12)

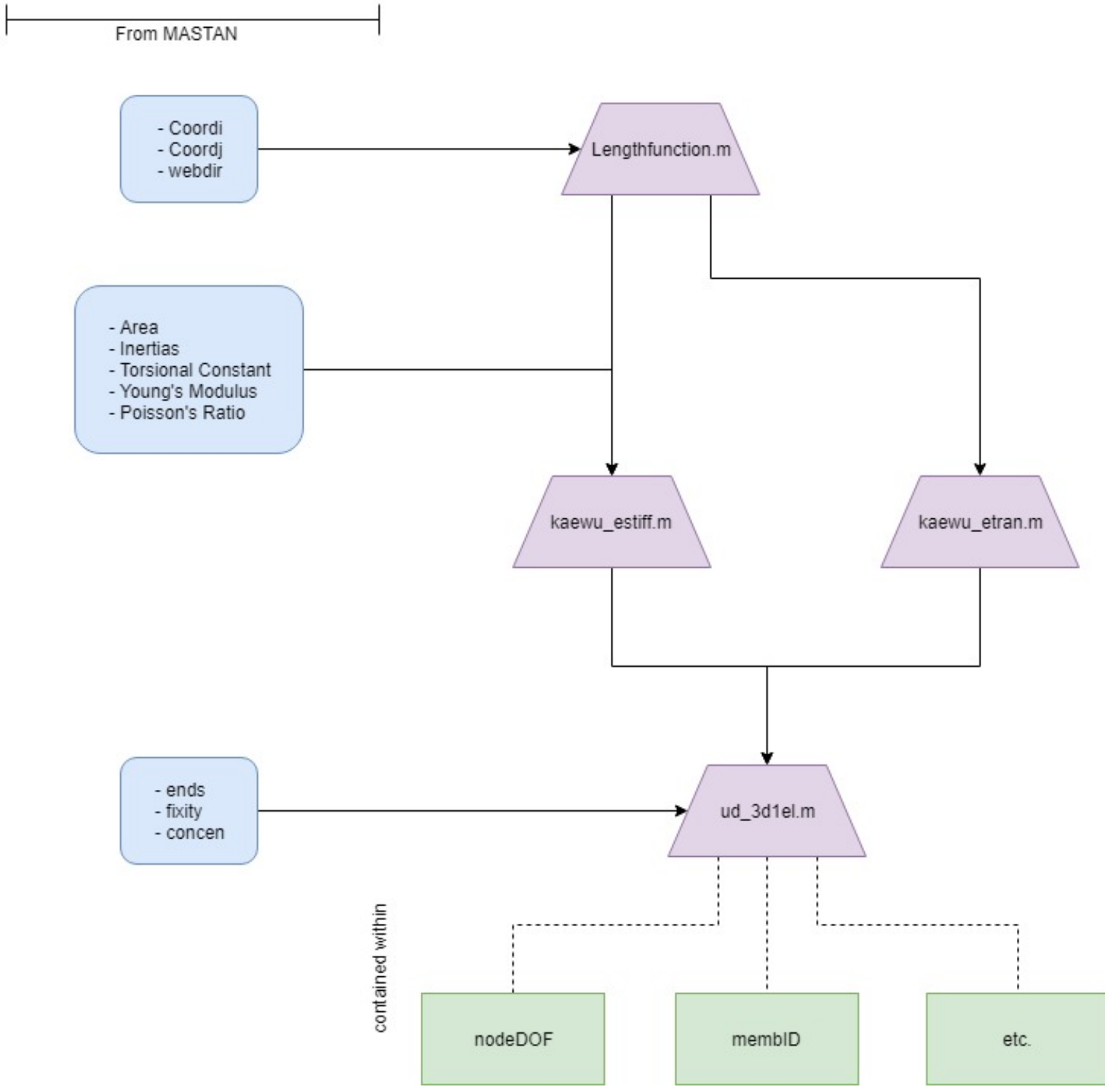
kaewu_etran.m

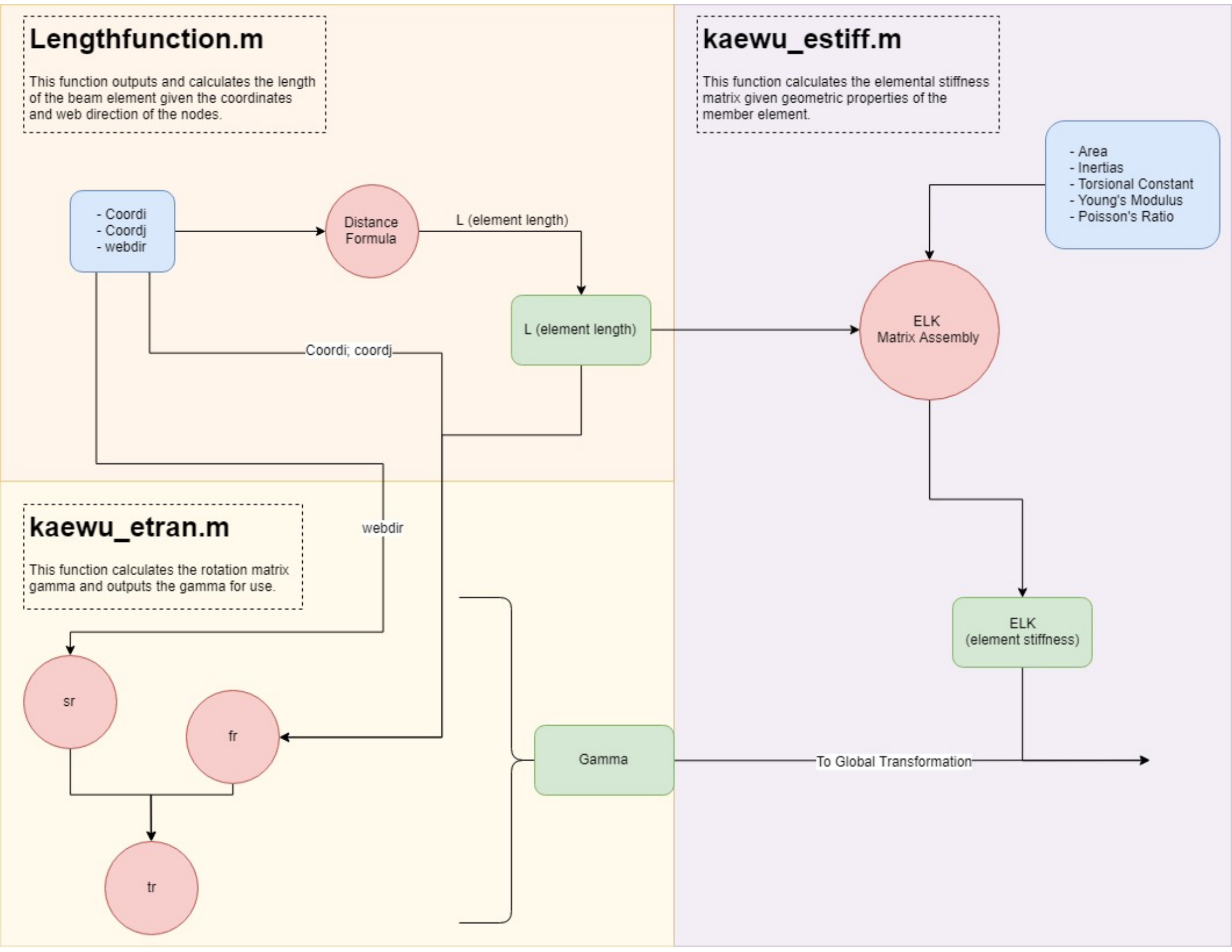
The angle of rotation of an element in global coordinates is calculated in a subroutine for each element as the main code loops over each member. This function is used to transform an element's local coordinate system to the global coordinate system.

Inputs:	coordi(ends(i,1))	coordj(ends(I,2))
	webdir(i,:)	L
Outputs:	gamma(12,12)	

CEE 235A: Functions Outline

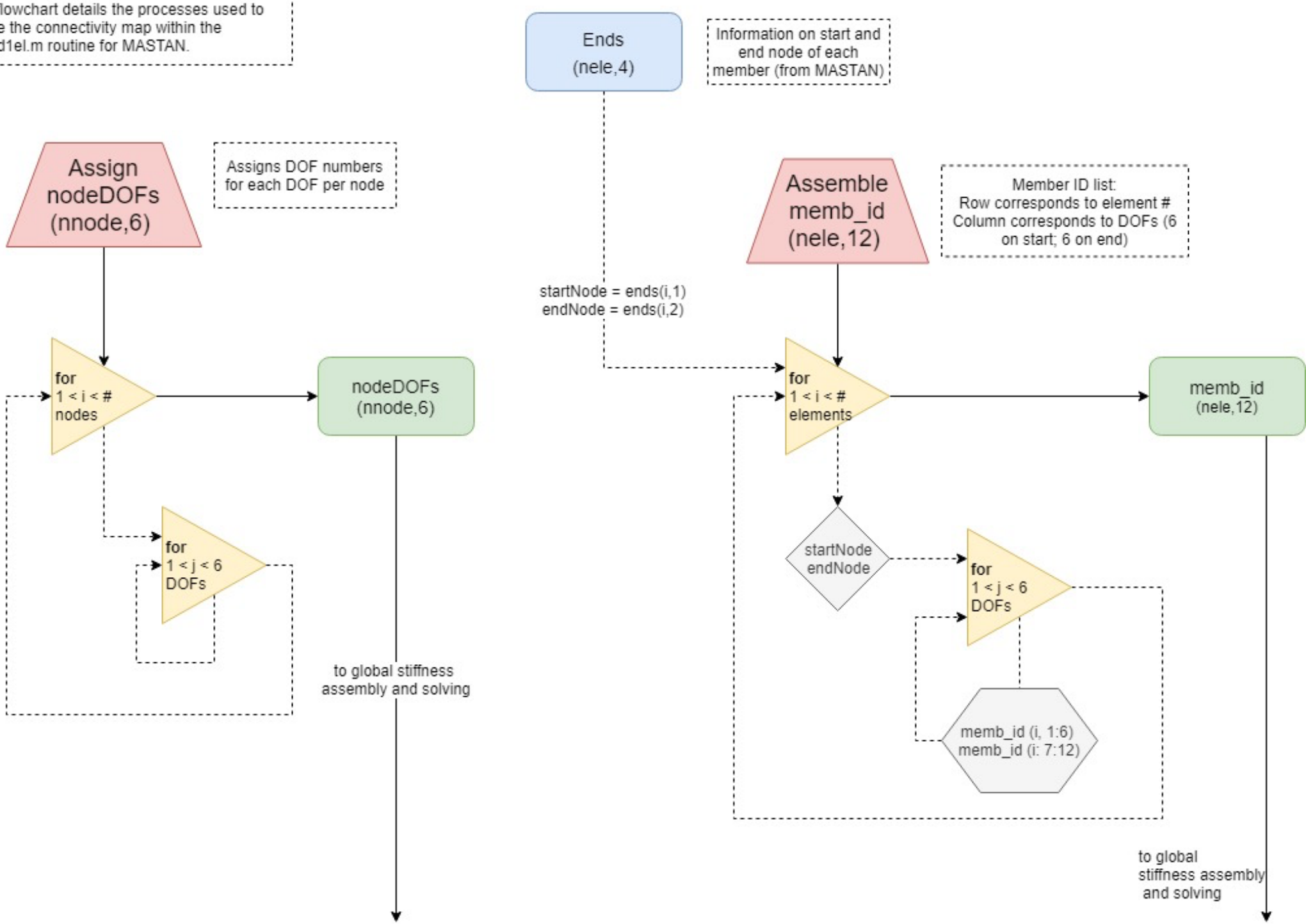
This flow chart shows the general structure of the subroutines and functions that will be used in the FEM project for a 3d element.




Lengthfunction.m
This function outputs and calculates the length of the beam element given the coordinates and web direction of the nodes.
kaewu_estiff.m
This function calculates the elemental stiffness matrix given geometric properties of the member element.
- Area
- Inertias
- Torsional Constant
- Young's Modulus
- Poisson's Ratio
- Coordi
- Coordj
- webdir
Distance Formula
L (element length)
L (element length)
ELK Matrix Assembly
Coordi; coordj
webdir
kaewu_etran.m
This function calculates the rotation matrix gamma and outputs the gamma for use.
ELK (element stiffness)
sr
fr
tr
Gamma
To Global Transformation

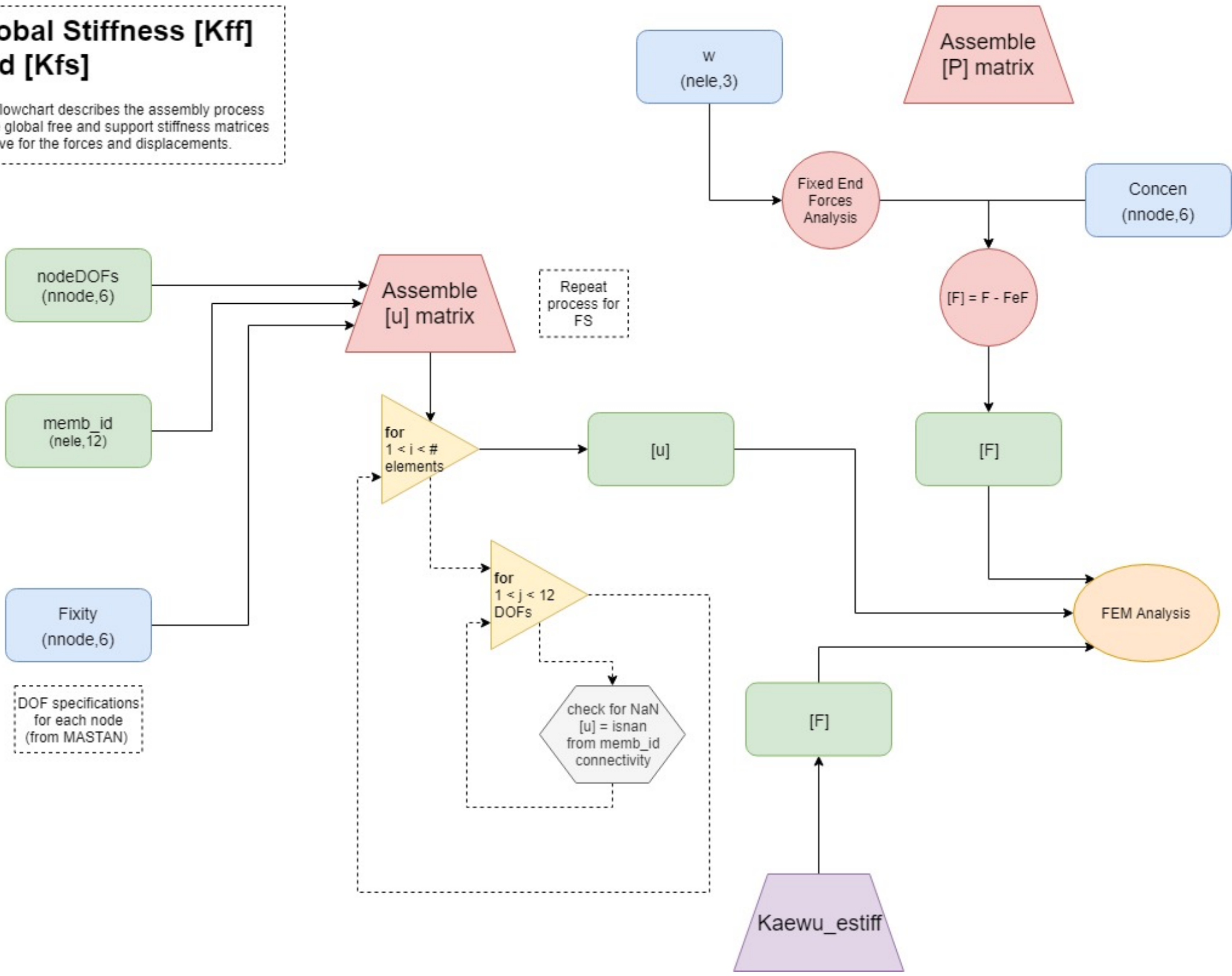
Connectivity Arrays

This flowchart details the processes used to create the connectivity map within the ud_3d1el.m routine for MASTAN.



Global Stiffness [Kff] and [Kfs]

This flowchart describes the assembly process of the global free and support stiffness matrices to solve for the forces and displacements.



```
function [DEFL,REACT,ELE_FOR,AFLAG] = ud_3d1el(...
        nnodes,coord,concen,fixity,nele,ends,A,Izz,Iyy,J,Cw,Zzz,Zyy,Ayy,Azz, ...
        E,v,Fy,YldSurf,Wt,webdir,beta_ang,w,thermal,truss,anatype);
%UD_3D1EL performs a user defined three-dimensional
% first-order elastic analysis of a structural system.
%

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Functions Called
%              < to be defined by the student >
%
%  Dictionary of Variables
%     Input Information:
%       nnodes         ==  total number of nodes
%       coord(i,1:3)   ==  node i's coordinates
%                            coord(i,1) = X coordinate
%                            coord(i,2) = Y coordinate
%                            coord(i,3) = Z coordinate
%       concen(i,1:6)  ==  concentrated loads for node i's 6 d.o.f.
%                            concen(i,1) = force in global X direction
%                            concen(i,2) = force in global Y direction
%                            concen(i,3) = force in global Z direction
%                            concen(i,4) = moment about global X axis
%                            concen(i,5) = moment about global Y axis
%                            concen(i,6) = moment about global Z axis
%       fixity(i,1:6)  ==  prescribed displacements for node i's 6 d.o.f.
%                          Note: A free d.o.f. will have a value of NaN
%                          and hence, you will find the Matlab function
%                          isnan very useful.
%                          Examples: If fixity(15,3) is set to NaN, then node 15's
%                                      Z-disp component is free;
%                                    If fixity(2,6) is set to 0.0, then node 2's
%                                      Z-rotation component is supported;
%                                    If fixity(5,2) is set to -2.1, then node 5's
%                                      Y-disp component is supported and defined
%                                      with a settlement of -2.1 units.
%                            fixity(i,1) = prescribed disp. in global X direction
%                            fixity(i,2) = prescribed disp. in global Y direction
%                            fixity(i,3) = prescribed disp. in global Z direction
%                            fixity(i,4) = prescribed rotation about global X axis
%                            fixity(i,5) = prescribed rotation about global Y axis
%                            fixity(i,6) = prescribed rotation about global Z axis
%       nele           ==  total number of elements
%       ends(i,1:14)   ==  element i's nodal information
%                            ends(i,1) = start node #
%                            ends(i,2) = finish node #
%                            ends(i,3) = flag to indicate whether or not flexural
%                            moments are released at start node.  ends(i,3)=0 both not
```

```
%                           released (rigid connection); ends(i,3)=1 both flexural
%                           moments are released (pinned connection); ends(i,3)=2
%                           at least one of the flexural moments are partially or ↙
fully
%                           released (see below for connection stiffness attributes)
%                           ends(i,4) = flag to indicate whether or not flexural
%                           moments are released at finish node.  ends(i,4)=0 both not
%                           released (rigid connection); ends(i,4)=1 both flexural
%                           moments are released (pinned connection); ends(i,4)=2
%                           at least one of the flexural moments are partially or ↙
fully
%                           released (see below for connection stiffness attributes)
%                           ends(i,5) = flag to indicate the degree of warping
%                           restraint at start node.  ends(i,5)=0 warping free;
%                           ends(i,5)=1 warping fixed; ends(i,5)=2 warping continuous
%                           ends(i,6) = flag to indicate the degree of warping
%                           restraint at finish node.  ends(i,6)=0 warping free;
%                           ends(i,6)=1 warping fixed; ends(i,6)=2 warping continuous
%                           ends(i,7) = rotational spring stiffness at the start
%                           node and about element i's local z-z axis.
%                           ends(i,8) = rotational spring stiffness at the start
%                           node and about element i's local y-y axis.
%                           ends(i,9) = rotational spring stiffness at the finish
%                           node and about element i's local z-z axis.
%                           ends(i,10) = rotational spring stiffness at the finish
%                           node and about element i's local y-y axis.
%                           ends(i,11) = connection moment capacity Mpz at the start
%                           node and about element i's local z-z axis.
%                           ends(i,12) = connection moment capacity Mpy at the start
%                           node and about element i's local y-y axis.
%                           ends(i,13) = connection moment capacity Mpz at the finish
%                           node and about element i's local z-z axis.
%                           ends(i,14) = connection moment capacity Mpy at the finish
%                           node and about element i's local y-y axis.
%       A(i)            ==  element i's cross sectional area
%       Izz(i)          ==  element i's moment of inertia about its local z-z axis
%       Iyy(i)          ==  element i's moment of inertia about its local y-y axis
%       J(i)            ==  element i's torsional constant
%       Cw(i)           ==  element i's warping constant
%       Zzz(i)          ==  element i's plastic section modulus about its local z-z axis
%       Zyy(i)          ==  element i's plastic section modulus about its local y-y axis
%       Ayy(i)          ==  element i's effective shear area along its local y-y axis
%       Azz(i)          ==  element i's effective shear area along its local z-z axis
%       E(i)            ==  element i's material elastic modulus, Young's Modulus
%       v(i)            ==  element i's material Poisson's ratio
%       Fy(i)           ==  element i's material yield strength
%       YldSurf(i)      ==  element i's yield surface maximum values
%                               YldSurf(i,1) = maximum P/Py value
%                               YldSurf(i,2) = maximum Mz/Mpz value
%                               YldSurf(i,3) = maximum My/Mpy value
```

```
%       Wt(i)          ==  element i's material weight density
%                          (Assume that gravity is directed in the negative global Y ↙
dir)
%       webdir(i,1:3)  ==  element i's unit web vector.  This is a unit vector
%                          that defines the element's local y-y axis with respect
%                          to the global coordinate system.  It is based on the
%                          structure's undeformed geometry.
%                              webdir(i,1) = x component of element's unit web vector
%                              webdir(i,2) = y component of element's unit web vector
%                              webdir(i,3) = z component of element's unit web vector
%                          NOTE: An element's 3x3 rotation matrix, [g], is constructed
%                          as follows: First, calculate a unit vector, x_vect, that
%                          describes the element's local x-axis. Second, take the
%                          cross product of x_vect and webdir(i,:) to obtain z_vect,
%                          i.e. z_vect = cross(x_vect,webdir(i,:)). Third, set z_vect
%                          to a unit vector, i.e. z_vect = z_vect/norm(z_vect).
%                          Finally, the first row of [g] is x_vect, its second row is
%                          webdir(i,:), and its third row is z_vect.
%       beta_ang(i)    ==  element i's web rotation angle.  These values are
%                          provided for those students who are required to calculate
%                          their own unit web vectors (see above).  It is based
%                          on the structure's undeformed geometry.
%                          Note:  MASTAN2 uses the following convention for
%                                 defining a member's default web orientation:
%                                 A vector defing the element's local y-axis
%                                 with respect to the global coordinate system
%                                 will have a positive component in the global
%                                 Y direction.  If the element's local x-axis,
%                                 its length axis, is aligned with the global Y
%                                 axis, then element's local y-axis is aligned
%                                 with global negative X axis.  After this initial
%                                 orientation, element i may be rotated about
%                                 its local x-axis by the amount defined by
%                                 its web rotation angle, beta_ang(i).  The
%                                 angle is in radians and assumes a right-hand
%                                 convention about the local x-axis which runs from
%                                 the element's start node to its finish node.
%       w(i,1:3)         ==  element i's uniform load which references its
%                            local coordinate system
%                              w(i,1) = x component of uniform load
%                              w(i,2) = y component of uniform load
%                              w(i,3) = z component of uniform load
%       thermal(i,1:4)   ==  element i's thermal strain effects which reference its
%                            local coordinate system
%                              thermal(i,1) = coefficient of thermal expansion
%                              thermal(i,2) = change in temperature at centroid
%                              thermal(i,3) = linear temperature gradient in local y- ↙
dir
%                                           = (T_up_y - T_btm_y) / depth_y
%                              thermal(i,4) = linear temperature gradient in local z- ↙
```

```
dir
%                                          = (T_up_z - T_btm_z) / width_z
%       truss             ==  flag to indicate if structure is a truss or not
%                               truss = 0   System is not a truss
%                               truss = 1   System is a truss
%       anatype           ==  flag to indicate which type of analysis is requested
%                               anatype = 1  First-Order Elastic
%                               anatype = 2  Second-Order Elastic
%                               anatype = 3  First-Order Inelastic
%                               anatype = 4  Second-Order Inelastic
%                               anatype = 5  Elastic Buckling (Eigenvalue)
%                               anatype = 6  Inelastic Buckling (Eigenvalue)
%=====================================================================
%     Local Information:
%              < to be defined by the student >
%
%     Output Information:
%       DEFL(i,1:6)       ==  node i's calculated 6 d.o.f. deflections
%                               DEFL(i,1) = displacement in X direction
%                               DEFL(i,2) = displacement in Y direction
%                               DEFL(i,3) = displacement in Z direction
%                               DEFL(i,4) = rotation about X direction
%                               DEFL(i,5) = rotation about Y direction
%                               DEFL(i,6) = rotation about Z direction
%       REACT(i,1:6)      ==  reactions for supported node i's 6 d.o.f.
%                               REACT(i,1) = force in X direction
%                               REACT(i,2) = force in Y direction
%                               REACT(i,3) = force in Z direction
%                               REACT(i,4) = moment about X direction
%                               REACT(i,5) = moment about Y direction
%                               REACT(i,6) = moment about Z direction
%       ELE_FOR(i,1:1?)   ==  element i's internal forces and moments
%                             Note: All values reference the element's local
%                                   coordinate system.
%                               ELE_FOR(i,1)  = x-force at start node
%                               ELE_FOR(i,2)  = y-force at start node
%                               ELE_FOR(i,3)  = z-force at start node
%                               ELE_FOR(i,4)  = x-moment at start node
%                               ELE_FOR(i,5)  = y-moment at start node
%                               ELE_FOR(i,6)  = z-moment at start node
%                               ELE_FOR(i,7)  = x-force at end node
%                               ELE_FOR(i,8)  = y-force at end node
%                               ELE_FOR(i,9)  = z-force at end node
%                               ELE_FOR(i,10) = x-moment at end node
%                               ELE_FOR(i,11) = y-moment at end node
%                               ELE_FOR(i,12) = z-moment at end node
%                             If you are not programming warping torsion, the ELE_FOR
%                             array needs to contain only 12 columns, i.e. ELE_FOR(i,1: ↙
12)
%                             For those programming warping torsion, the bimoments and
```

```
%                             rates of twist should be stored as follows.
%                               ELE_FOR(i,13) = bimoment at start node
%                               ELE_FOR(i,14) = bimoment at end node
%                               ELE_FOR(i,15) = rate of twist at start node
%                               ELE_FOR(i,16) = rate of twist at end node
%       AFLAG             ==  logical flag to indicate if a successful
%                             analysis has been completed
%                               AFLAG = 1     Successful
%                               AFLAG = 0     Unstable Structure
%                               AFLAG = inf   No analysis code available
%
%
%       Version 1.0/Student's Initials/Date of Modification
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%
%  Start by defining all output arrays to be empty
    DEFL=[]; REACT=[]; ELE_FOR=[];
    AFLAG = inf;
%
%  STUDENT NOTE:
%     In order for this routine to become fully active AFLAG
%     must be changed.
%
%
%  Student's code starts here...
%
%================================================================================
clc; tic
%
% generate the memb_id array containing DOF #s for each end of the member
%   row # i is the member ID
%   column # 1~6 are the DOFs corresponding to end 1 of the member
%   column # 7~12 are the DOFs corresponding to end 2 of the member
% outer loop for # of elements
%
%----------------------------------------------------------------------
% generate the DOF mapping for each node
%----------------------------------------------------------------------
    % outer loop for total number of nodes
    for i = 1:nnodes
        % inner loop for DOF # per node
        for j=1:6
            node_dofs(i,j) = (i-1)*6 +j;
        end
    end
    % output node_dofs array to check
    node_dofs;
    w;
```

```
%
% generate the memb_id array
%
    % outer loop for number of elements
    for i = 1:nele
        % grab node #s from the ends array
        startNodeNumber = ends(i,1);
        endNodeNumber = ends(i,2);

        % inner loop for the 6 DOFs per node
        for j = 1 : 6
            memb_id(i,j) = (startNodeNumber-1)*6+j;
            memb_id(i,j+6) = (endNodeNumber-1)*6+j;
        end
    end
    % output memb_id array to check
    memb_id;
%
%---------------------------------------------------------------------
% generate the concentrated load vector for applied nodal loads
%---------------------------------------------------------------------
    % outer loop across number of nodes
    for i = 1:nnodes
        % there are 6 DOFs per node
        for j = 1:6
            appliedNodalLoads(node_dofs(i,j),1)= concen(i,j) ;
        end
    end
    % output applied nodal load vector to check
    appliedNodalLoads;
%
%---------------------------------------------------------------------
% Convert the fixity array to a column vector
%---------------------------------------------------------------------
    fixity
    D = transpose(fixity);
    D = D(:);           %creating a column vector

    % find free degrees of freedom from the fixity array; isnan is free
    freeDOF = find(isnan(D));

    % find supports from fixity array; 0 values is supported
    supportDOF = find(D==0);

    % find prescribed displacements; non-zero values
    displacedDOF = find(D~=0 & ~isnan(D));
%
%---------------------------------------------------------------------
% Creating Array of Nodal Loads Corresponding to Fixed-End-Forces
%---------------------------------------------------------------------
```

```
    % initialize empty array of zeros for Fixed-end-forces
    FEF = zeros(6*nnodes,1);


    for  i = 1:nele
        % Use Length Function to compute the length of each member;
        %       which will be used to calculated the fixed end forces
        coordi = coord(ends(i,1),:);
        coordj = coord(ends(i,2),:);
        L = lengthfunction(coordi, coordj);

        % calculate the local fixed-end-forces for bi-directional loads
        memberLocalFEF = kaewu_computeMemberFEFs(w(i,:),L);
        memberLocalFEF = transpose(memberLocalFEF);

        % Compute the transformation matrix for element i
        gamma = kaewu_etran(coord(ends(i,1),:),coord(ends(i,2),:),webdir(i,:),L);

        % Transform the fixed-end-forces from local to global coordinates
        memberglobalFEF = transpose(gamma)* memberLocalFEF;
        %memberglobalFEF = transpose(memberglobalFEF)
        % commented out since memberLocalFEF is [1x12]

        % Populate the vector of fixed-end-forces at each degree of freedom
        FEF(memb_id(i,:),1) = memberglobalFEF + FEF(memb_id(i,:),1);
    end
%
%---------------------------------------------------------------------
% Construct the structure stiffness matrix
%---------------------------------------------------------------------
%   % initialize zeros for sturcture stiffness matrix
    k_structure_global = zeros(6*nnodes, 6*nnodes);

    % loop over number of elements in structure
    for i = 1:nele
        % compute stiffness for element in local coords
        coordi = coord(ends(i,1),:);
        coordj = coord(ends(i,2),:);
        L = lengthfunction(coordi, coordj);
        k_ele_local = kaewu_estiff(A(i),Izz(i),Iyy(i),J(i),Ayy(i),Azz(i),E(i),v(i),L);

        % compute transformation matrix for element
        gamma = kaewu_etran(coordi, coordj,  webdir(i,:),L);

        % compute stiffness matrix in global coordinates
        k_ele_global = transpose(gamma)*k_ele_local*gamma;

        % populate global structure stiffness matrix
        k_structure_global(memb_id(i,:),memb_id(i,:)) = k_ele_global + ↙
k_structure_global(memb_id(i,:) , memb_id(i,:));
```

```
    end
    k_structure_global
%
%-----------------------------------------------------------------------
% Compute the appliedNodalLoads with FeF's applied
%-----------------------------------------------------------------------
    % subtract the FeF forces from the applied nodal loads
    %appliedNodalLoads = appliedNodalLoads - FEF ;
    % we can do this later since we will reuse FEF
%
%-----------------------------------------------------------------------
% Extract the Kff, Kfn, etc. from the global structure stiffness K
%-----------------------------------------------------------------------

    % K values corresponding to free-free DOFs
    Kff = k_structure_global(freeDOF, freeDOF)

    % K values corresponding to free-displaced DOFs
    Kfn = k_structure_global(freeDOF, displacedDOF)

    % K values corresponding to supported-free
    Ksf = k_structure_global(supportDOF, freeDOF)

    % K values corresponding to displaced-displaced
    Knn = k_structure_global(displacedDOF, displacedDOF)

    % K values corresponding to displaced-supported
    Ksn = k_structure_global(supportDOF, displacedDOF)


%-----------------------------------------------------------------------
% Solve for displcaments at free degrees of freedom
%-----------------------------------------------------------------------

    % extract the applied loads at the free DOFs
    Pf = appliedNodalLoads(freeDOF)
    R(freeDOF) = Pf

    % solve for nodal displacements at freeDOF
    D(freeDOF) = inv(Kff)*(Pf-FEF(freeDOF)-Kfn*D(displacedDOF))

    % reaction at supports
    R(supportDOF) = Ksf*D(freeDOF) + Ksn*D(displacedDOF) + FEF(supportDOF)

    %% check the transpose part here...i had to transpose since 7x1 and 7x1, but what ↙
if
    % we have different number of displaced settlements?
    % reaction at the displaced settlements
    R(displacedDOF) = transpose(Kfn)*D(freeDOF) + Knn*D(displacedDOF) + FEF ↙
(displacedDOF)
```

```
    %
    %%



%Plug back in the displacements at the free dofs in the nodes displacement
%vector D


%---------------------------------------------------------------------
% Solve and Compute the member forces of each element
%---------------------------------------------------------------------
    % loop over the number of elements in the entire structure
    for i = 1:nele
        % compute the stiffness matrix for element i in local coords
        coordi = coord(ends(i,1),:);
        coordj = coord(ends(i,2),:);
        L = lengthfunction(coordi, coordj);
        k_ele_local = kaewu_estiff(A(i), Izz(i), Iyy(i), J(i), Ayy(i), Azz(i), E(i), v ↙
(i), L);


        % compute the transformation matrix for element i
        gamma = kaewu_etran(coordi, coordj, webdir(i,:),L);

        % extract the vector of global displacements asosciated with
        % element i

        Dele_global = D(memb_id(i,:),1);
        Dele_local = gamma*Dele_global;

        % Compute the fixed-end-forces assoaciated with element i
        memberLocalFEF = kaewu_computeMemberFEFs(w(i,:),L);
        memberLocalFEF = transpose(memberLocalFEF);

        % Compute local member forces
        localMemberForces = k_ele_local*Dele_local + memberLocalFEF;

        % Insert the member forces for element i into the ELE_FOR array to
        % be read by MASTAN2
        ELE_FOR(i,:) = transpose(localMemberForces);
    end
%
%---------------------------------------------------------------------
% Convert the displacement vector at freeDOF Df to the [DEFL] array that is
% read by MASTAN2
%---------------------------------------------------------------------

    % output the reactions at the nodes
```

```
    REACT = reshape(R,6,nnodes);
    REACT = transpose(REACT)

    % output deflections at the nodes
    DEFL = reshape(D,6,nnodes);
    DEFL = transpose(DEFL)



    ELE_FOR
%-------------------------------------------------------------------------
% Change AFLAG for successful run
%-------------------------------------------------------------------------
    AFLAG = 1;
    toc
%=========================================================================
%
%  Good luck CE Student!!!
%
```

```
% length function
function [L] = lengthfunction(coordi, coordj)

%this function calculates the length given the coordinates of node i and j
L = sqrt(((coordi(1)-coordj(1))^2 + (coordi(2)-coordj(2))^2 +(coordj(3)-coordi(3))^2));


end
```

```
function [gamma] = kaewu_etran(coordi, coordj, webdir,L)
% returns the rotation matrix gamma given input L from other function%


fr = [((coordj(1)-coordi(1))/L), ((coordj(2)-coordi(2))/L), ((coordj(3)-coordi(3))/L)];
sr = [webdir(1),webdir(2),webdir(3)];
tr = (cross(fr,sr));


gamma = [ fr 0 0 0 0 0 0 0 0 0;
          sr 0 0 0 0 0 0 0 0 0;
          tr 0 0 0 0 0 0 0 0 0;
          0 0 0 fr 0 0 0 0 0 0;
          0 0 0 sr 0 0 0 0 0 0;
          0 0 0 tr 0 0 0 0 0 0;
          0 0 0 0 0 0 fr 0 0 0;
          0 0 0 0 0 0 sr 0 0 0;
          0 0 0 0 0 0 tr 0 0 0;
          0 0 0 0 0 0 0 0 0 fr;
          0 0 0 0 0 0 0 0 0 sr;
          0 0 0 0 0 0 0 0 0 tr;];


end
```

```
function [elk] = kaewu_estiff(A, Izz, Iyy, J, Ayy, Azz, E, v, L)

G = (E/(2*(1+v)));
n = ((Izz)/(A*(5/6)*G));
coff = ((Izz)/(L^3/(12) +L*n));
%The Above variables are used to calculate the shear coefficient
%to change the affected indices.



elk =            E*[(A/L) 0 0 0 0 0 -(A/L) 0 0 0 0 0;
                  0 coff 0 0 0 (coff*L/2) 0 -(coff) 0 0 0 (coff*L/2);
                  0 0 (12*Iyy/L^3) 0 -(6*Iyy*1/L^2) 0 0 0 -(12*Iyy/L^3) 0 -(6*Iyy/L^2) ↙
0;
                  0 0 0 (J/(2*(1+v)*L)) 0 0 0 0 0 -(J/(2*(1+v)*L)) 0 0 ;
                  0 0 -(6*Iyy*1/L^2) 0 (4*Iyy/L) 0 0 0 -(6*Iyy*1/L^2) 0 (2*Iyy/L) 0 ;
                  0 (coff*L/2) 0 0 0 (coff*((L^2)/3 +n)) 0 -((coff*L/2)) 0 0 0 coff* ↙
((L^2)/6 - n);
                  -(A/L) 0 0 0 0 0 (A/L) 0 0 0 0 0;
                  0 -(coff) 0 0 0 -(coff*L/2) 0 (coff) 0 0 0 -(coff*L/2);
                  0 0 -(12*Iyy/L^3) 0 (6*Iyy*1/L^2) 0 0 0 (12*Iyy/L^3) 0 (6*Iyy*1/L^2) ↙
0;
                  0 0 0 -(J/(2*(1+v)*L)) 0 0 0 0 0 (J/(2*(1+v)*L)) 0 0 ;
                  0 0 -(6*Iyy*1/L^2) 0 (2*Iyy/L) 0 0 0 (6*Iyy*1/L^2) 0 (4*Iyy/L) 0;
                  0 (coff*L/2) 0 0 0 coff*(L^2/6 - n) 0 -((coff*L/2)) 0 0 0 coff*(L^2/3 ↙
+n)] ;




end
```

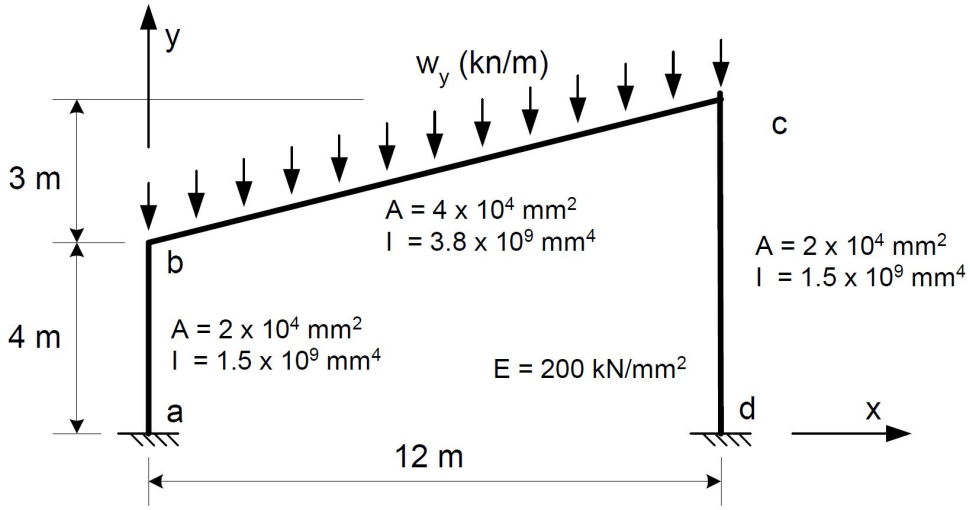
```
function [FeF] = kaewu_computeMemberFEFs(w,L)

% compute FeF using the same DOF numbering as memb_id
%   dx1, dy1, dz1, thetax1, thetay1, thetaz1, ...
%   dx2, dy2, dz2, thetax2, thetay2, thetaz2

% print out distributed load to check
w ;

% x-component distributed load
w_x = w(1);
% y-component distributed load
w_y = w(2);
% z-component distributed load
w_z = w(3);


% Rx due to w_x; Mx is 0 since x along element
Rx = (w_x*L/2);
Mx = 0;

% Ry due to w_y; My due to w_z
Ry = (w_y*L/2);
My = w_z*L^2/12;

% Rz due to w_z; Mz due to w_y
Rz = (w_z*L/2);
Mz = w_y*L^2/12;

% node i has reaction moments opposite in sign of load
% node j has reaction moments same sign of load if
% counterclockwise positive and tensile loads positive
FeF_i = [-Rx, -Ry, -Rz, 0, -My, -Mz];
FeF_j = [-Rx, -Ry, -Rz, 0,  My,  Mz];

FeF = [ FeF_i, FeF_j];


% note that FeF have a positive rotation at the 2nd node of a member


end
```

Summary Tables of Verification Problems

Verification Problem 1

The following verification table shows the results of the FEM analysis for problem 1, which is shown in the picture below. Note that the errors are small and only a fraction of a percent, which can likely be attributed to numerical solver errors.

Verification Problem 1a:



Notes:

1) The load W_y = 15 kN/m is a vertical distributed load along the length of the member, which you will need to convert to equivalent amounts of distributed load in the local x' and y' axis of the member.

Report the following information:

- Deflections at point b (Δx, Δy, θz)
- Reactions at point a (Fx, Fy, Mz)
- Sketch of bending moment diagram showing numeric values at member ends and midspan of b-c.

Verification for Problem 1a

Deflections

Point b	MASTAN	units	Ours	units	Error
delta_x	0.745	mm	0.7448	mm	-0.03%
delta_y	-0.09817	mm	-0.09817	mm	0.00%
theta_z	-0.000622	rad	-0.0006226	rad	0.10%

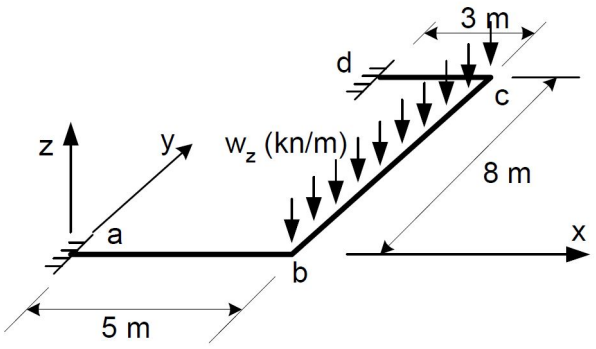
Reactions

Point a	MASTAN	units	Ours	units	Error
Fx	28.13	kN	28.13	kN	0.00%
Fy	98.17	kN	98.17	kN	0.00%
Mz	-9572	kN-mm	-9554	kN-mm	-0.19%
Point d					
Fx	-28.14	kN	-28.13	kN	-0.04%
Fy	87.37	kN	87.37	kN	0.00%
Mz	7.44E+04	kN-mm	7.44E+04	kN-mm	-0.03%

Verification Problem 2

The verification table for problem 2 is shown below. A diagram of the structure is included in the picture. Note the complete agreement of our results with that of MASTAN.

Verification Problem 2:



Notes:

1) The structure consists of a horizontal grid of rectangular tubular members measuring 100 x 300 mm square. The members are all oriented with their tall dimension parallel to the global z-axis (vertical direction). The tubular members have the following properties: A = 11,000 mm^2, Imajor = 1.06 x 10^8 mm^4, Iminor = 1.74 x 10^7 mm^4 , J = 5.29 x 10^7 mm^4
2) Members are steel with E = 200 kN/mm^2 and $\nu = 0.3$.
3) The load W_z = 5 kN/m is a vertical distributed load along the length of the member.

Report the following information:

- Deflections at point b (Δx, Δy, Δz, θz, θy, θz)
- Reactions at point a (Fx, Fy, Fz, Mx, My, Mz)
- Value of torsion (Mx') in member a-b.
- Sketch diagram of major axis bending for each member with key numerical values indicated.

Verification for Problem 2

Deflections

Point b	MASTAN	units	Ours	units	
delta_x	0	mm	0	mm	#DIV/0!
delta_y	0	mm	0	mm	#DIV/0!
delta_z	-35.5	mm	-35.5	mm	0.00%
theta_x	-0.001078	rad	-0.001078	rad	0.00%
theta_y	-0.01048	rad	-0.01048	rad	0.00%
theta_z	0	rad	0	rad	#DIV/0!

Reactions

Point a	MASTAN	units	Ours	units	
Fx	0	kN	0	kN	#DIV/0!
Fy	0	kN	0	kN	#DIV/0!
Fz	18.92	kN	18.92	kN	0.00%
Mx	877.7	kN-mm	877.7	kN-mm	0.00%
My	9.17E+04	kN-mm	9.17E+04	kN-mm	0.00%
Mz	0	kN-mm	0	kN-mm	#DIV/0!

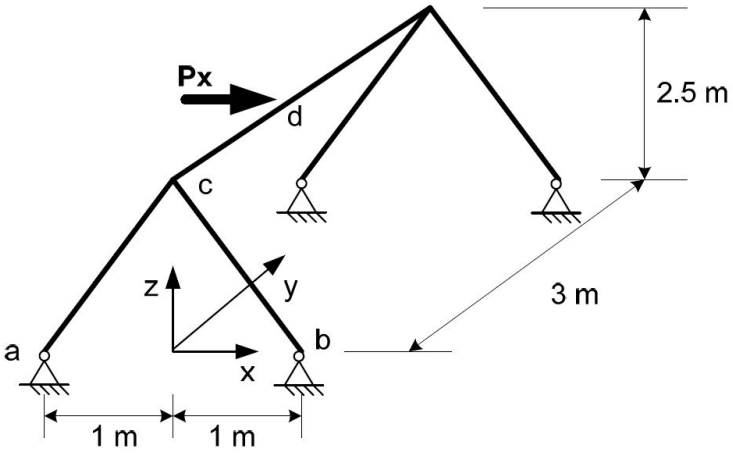
Torsion

Member	MASTAN	units	Ours	units	
ab	877.7	kN	877.7	kN	0.00%

Verification Problem 3

The verification table for Problem 3 is shown below. There are many discrepancies with this model. As such, we assumed that there was a bug in the code. Unfortunately, given the exact results of the other verification problems, and the rebuilding of MASTAN models numerous times, we determined that the issue was not in the code. This conclusion was reached when code modifications created worse results in other verification problems without mitigating the error in problem 3. It may be possible that there are unit issues in the MASTAN model or coordinate issues that have not been resolved. Nevertheless, with the accuracy of other verification problems, we determined that this problem is likely an outlier that would be investigated given more time; but better off left alone for the purpose of submitting this project.

Verification Problem 3 – Swing Set:



Notes:

1) The structure is built with round 75 mm diameter tubular members have the following properties: A = 1,430 mm^2, I = 1.26 x 10^6 mm^4, J = 2.52 x 10^6 mm^4
2) Members are steel with E = 200 kN/mm^2 and ν = 0.3.
3) The load P_x = - 4.5 kN

Report the following information:

- Deflections at point d (Δx, Δy, Δz, θz, θy, θz)
- Reactions at points a and b (Fx, Fy, Fz, Mx, My, Mz)
- Axial forces in members a-c, c-b, and c-d.

Verification for Problem 3

Deflections

Point d	MASTAN	units	Ours	units	
delta_x	-5.789	mm	-2.545	mm	-56.04%
delta_y	0.00E+00	mm	0.00E+00	mm	#DIV/0!
delta_z	0.00E+00	mm	0.00E+00	mm	#DIV/0!
theta_x	0.00E+00	rad	0.00E+00	rad	#DIV/0!
theta_y	2.65E-05	rad	2.65E-05	rad	0.00%
theta_z	0.00E+00	rad	1.09E-03	rad	#DIV/0!

Reactions

Point a	MASTAN	units	Ours	units	
Fx	1.125	kN	1.53	kN	36.00%
Fy	-0.3306	kN	0.1848	kN	-155.90%
Fz	2.812	kN	3.823	kN	35.95%
Mx	-381.4	kN-mm	213.2	kN-mm	-155.90%
My	0.00E+00	kN-mm	0	kN-mm	#DIV/0!
Mz	-154.7	kN-mm	86.47	kN-mm	-155.90%

Point b	MASTAN	units	Ours	units	
Fx	1.125	kN	1.53	kN	36.00%
Fy	-0.3306	kN	-0.1848	kN	-44.10%
Fz	-2.812	kN	-3.823	kN	35.95%
Mx	381.4	kN-mm	-213.2	kN-mm	-155.90%
My	0.00E+00	kN-mm	0	kN-mm	#DIV/0!
Mz	-154.7	kN-mm	86.47	kN-mm	-155.90%

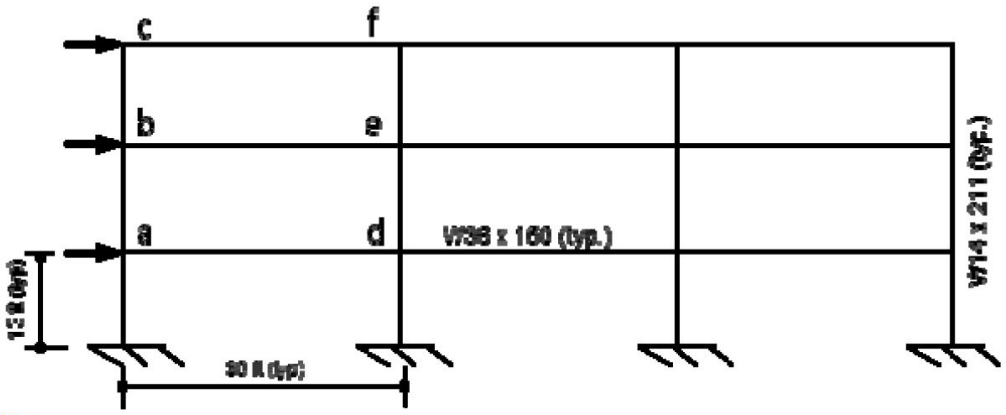
Axial

Member	MASTAN	units	Ours	units	
ac	-3.029	kN	-4.118	kN	35.95%
cb	3.029	kN	4.118	kN	35.95%
cd	0.00E+00	kN	0	kN	#DIV/0!

Verification Problem 4

The following table lists the verification for problem 4. While this is not as well matched as problem was, the results are still close enough to assume acceptable bounds.

Verification Problem 4a:



Notes:

1) Members have the following properties:
   W36 x 150: A=44.2 in^2, I = 9,040 in^4, A_{web} = 22.4 in^2
   W14 x 211: A=62.0 in^2, I = 2,660 in^4, A_{web} = 15.7 in^2
2) Members are steel with E = 30,000 k/in^2 and ν = 0.3.
3) The applied lateral load at each floor is Px = 9.5 kips
4) Base your analysis on centerline dimensions (i.e., ignoring finite joint size effects).

Perform two lateral load analyses, one in which shear deformations are included and one in which they are excluded. Report the following information for each analysis

- Lateral deflections at each floor level (Δx_a, Δx_b, Δx_c)
- The maximum moments in column a-d and beam b-e.
- What is the percentage change in lateral deflections due to the shear deformations?
- What is the percentage change in the *maximum* beam and column moments due to shear deformations?

Verification Problem 4b (Extra Credit):

Repeat Verification Problem 4a for the case where both ends of beams a-d, b-e and c-f are flexurally released.

Verification for Problem 4a

With Shear Deformation

Deflections

Lateral	MASTAN	units	Ours	units	
delta_x3 Roof	0.1001	mm	0.10892	mm	8.81%
delta_x2 3F	0.08656	mm	0.08655	mm	-0.01%
delta_x1 2F	0.04440	mm	0.044436	mm	0.08%

Max Moment	MASTAN	units	Ours	units	
col ab	324.00	kN-mm	324.6	kN-mm	0.19%
beam bd	431.8	kN-mm	429.5	kN-mm	-0.53%

Without Shear Deformation

Deflections

Lateral	MASTAN	units	Ours	units	
delta_x3 Roof	0.09695	mm	0.09695	mm	0.00%
delta_x2 3F	0.07769	mm	0.07769	mm	0.00%
delta_x1 2F	0.04091	mm	0.04091	mm	0.00%

Max Moment	MASTAN	units	Ours	units	
col ab	316.50	kN-mm	316.2	kN-mm	-0.09%
beam bd	325.7	kN-mm	362.5	kN-mm	11.30%

Percentage Differences

Lateral	MASTAN	Ours
delta_x3 Roof	3.15%	10.99%
delta_x2 3F	10.25%	10.24%
delta_x1 2F	7.86%	7.94%

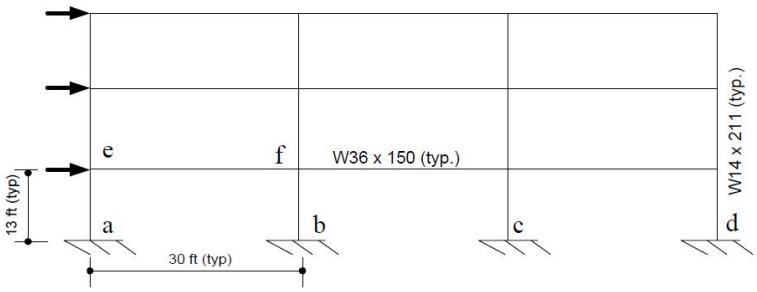
Max Moment		
col ab	2.31%	0.025878004
beam bd	24.57%	0.155995343

Verification of Problem 5

The verification table for Problem 5 shows an exact match in the model with our finite element code. This confirms the previous hypothesis that the code is correct and the results match; and problem 3 is an outlier.

CEE 235A Advanced Structural Analysis **December 5, 2018**
Programming Project Verification Problems

Verification Problem 5:



Notes:
1) This is the same structure as for Problem 4.
2) For this problem, do NOT include member shear deformations.

Perform an analysis where you apply a vertical settlement of Δ = -1 inch to the support at point b. Report the following information from this analysis.

- Base reactions at points a, b, c, and d (Fx, Fy, M)
- Shear and moments in beam e-f (V, Me, Mf).

Verification for Problem 5

Reactions

Point a	**MASTAN**	**units**	**Ours**	**units**	
Fx	22.76	kips	22.76	kips	0.00%
Fy	125	kips	125	kips	0.00%
Mz	-844.6	kips-in.	-844.6	kips-in.	0.00%
Point b	**MASTAN**	**units**	**Ours**	**units**	
Fx	-11.6	kips	-11.9	kips	2.59%
Fy	-285.2	kips	-285.2	kips	0.00%
Mz	982.2	kips-in.	982.2	kips-in.	0.00%
Point c	**MASTAN**	**units**	**Ours**	**units**	
Fx	-34.1	kips	-34.1	kips	0.00%
Fy	178.8	kips	178.8	kips	0.00%
Mz	2160	kips-in.	2160	kips-in.	0.00%
Point d	**MASTAN**	**units**	**Ours**	**units**	
Fx	-5.27	kips	-5.27	kips	0.00%
Fy	-18.61	kips	-18.61	kips	0.00%
Mz	654	kips-in.	654	kips-in.	0.00%
member ef	**MASTAN**	**units**	**Ours**	**units**	
v	43.17	kips	43.17	kips	0.00%
Me	-6480	kips	-6480	kips	0.00%
Mf	9062	kips-in.	9062	kips-in.	0.00%

Self – Assessment

Required Features

The program is modular and adheres to all the requirements of the project. Four out of five verification problems were tested to high degrees of accuracy. The one problem that was not accurate was modified numerous times. Modifications only made things worse for the other problems. We determined it was better to properly verify four out of five problems than none out of five.

Code Organization

The code is well documented with easy to follow comments. Sub-routines refer to functions that are called multiple times, which are not in the main code. The first page of this report shows an easy to follow outline that guides a reader into the details of the code, which are further explained in the comments throughout the code.

Project Write Up

The write up is a simple write up that should be easy to follow. The flow charts are simplified but convey the main idea. See above regarding verification problems.

Responsibilities

Interim 1

Khalid Alsadhan performed the tasks while Elmer Wu checked the code by using Professor Burton's Excel Sheet.

Interim 2

Tasks were evenly divided. Khalid wrote the required code while Elmer created the flow chart.

Interim 3

Elmer wrote the required code while Khalid built and tested MASTAN models on the code.

Final Submission

Both spent time finalizing the code and creating the report. Elmer re-wrote some of the code to make it more efficient while Khalid built the verification MASTAN models and sent it to Elmer to use the code to check the results.

CEE 235A Programming Project **DATE: _____**

Students Name: _____

Instructions to Students: *Include this sheet, along with the accompanying data sheets, with your project submission. Students should fill in the self-assessment parts of the form based on an objective review of your own teams work.*

Project Requirements:

(a) Functionality of the standard 1st-order elastic analysis routines in MASTAN2
(b) A short written description of program summarizing its features and the overall structure of the program, including a simple flow-chart of how the program works.
(c) A printed listing of your program. We expect that the main program file ***ud_3d1el.m*** will include comment statements that define and describe all the main variables and functions in your program. Each sub-function should also include appropriate comment statements.
(d) An electronic copy of your source code (we may run your program to check that it really works).
(e) A summary of results from the example/verification problems.

Grading:

[50 %] Required Features & Run Correctly – Does the program possess all of the required features and been demonstrated to run correctly? If not, briefly explain what the problems are and what steps were taken to try and resolve the problems.

STUDENT ASSESSMENT:

Student thoughts contained in the report

TA/PROFESSOR ASSESSMENT **(Score _____/50%)**

[30 %] Code Organization and Documentation: Is the source code well organized and well documented? Are all of the variables defined in the main program and the functions? Are the functions and their purpose defined in the source code comments? Is the code efficient in terms of run time and storage operations?

STUDENT ASSESSMENT:

Student thoughts contained in the report

TA/PROFESSOR ASSESSMENT **(_____/30%)**

[20%] Project Write-up: Is the project report well organized and complete? Is the program flow chart accurate and informative in understanding the code? Is the program overview and critique informative? Are all the verification problems complete?

STUDENT ASSESSMENT:

Student thoughts contained in the report.

TA/PROFESSOR ASSESSMENT **(Score _____/20%)**

Final Grade (by Professor): _____/100%