ME-7120 Finite Element Method Applications
Project 1

Sagar Sangle, Joel Summerfield & Andy Turner

Course: FEM
Section: ME-7120-01
Prof: Dr. Slater

Submitted: October 28th, 2016

Table of Contents

I. Nomenclature

δ	=	Displacement
P	=	Load
E	=	Young’s Modulus of Elasticity
I	=	Mass Moment of Inertia
D	=	Deflection
x	=	Deflection in the x-direction
y	=	Deflection in the y-direction
z	=	Deflection in the z-direction

II. Project Description

1. Non-Tapered Beam:
    a.) WFEM Analysis
    b.) ANSYS Analysis
    c.) Closed Form Solution

2. Rotated Non-Tapered Beam
    a.) Use WFEM to show transformation matrix is correct

3. Tapered Beam:
    a.) WFEM Analysis
    b.) ANSYS Analysis

4. Complex Truss
    a.) WFEM Analysis
    b.) ANSYS Analysis

III. Results & Interpretations

1. Non-Tapered Beam

WFEM and ANSYS were used to analyze a non-tapered, 2 noded, 3D beam element. All of the code that was used for this project can be found in the Appendix section of this report and/or the final submission folder. Figure 1 below shows the non-tapered beam that was analyzed. The boundary and loading conditions are shown as well as the geometry of the beam.

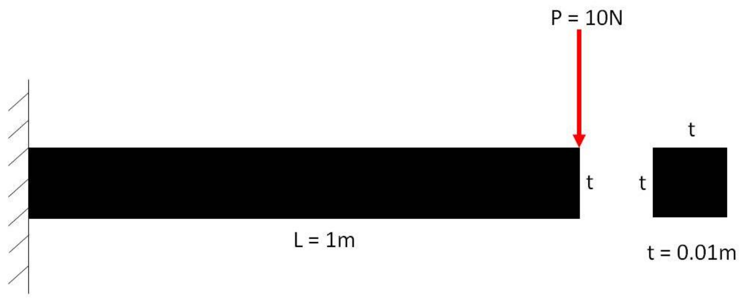

Figure 1: *Non-Tapered Beam*

Mesh convergence studies were conducted in both WFEM and ANSYS. Figures 2 and 3 show the non-tapered beam using WFEM with 1 element and 50 elements, respectively. Figures 4 and 5 show the non-tapered beam using ANSYS with 1 element and 50 elements, respectively. All the plots shown include both the un-deformed and deformed shapes. The results obtained by WFEM and ANSYS are shown in Tables 1 and 2 below , respectively. The natural frequencies for each mode were found using WFEM and are also shown in Table 1. To validate both methods of analysis, a closed form solution was calculated to compare to the converged results. Equations (1-3) show how the closed form solution for a non-tapered beam is obtained.

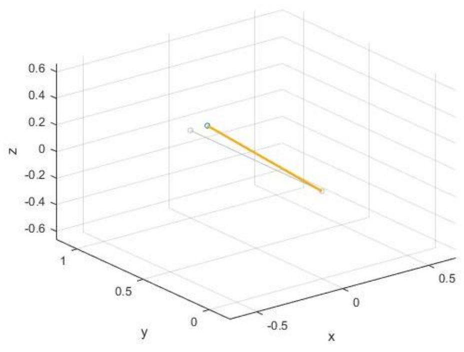

Figure 2: *Non-Tapered Beam in WFEM using 1 element*

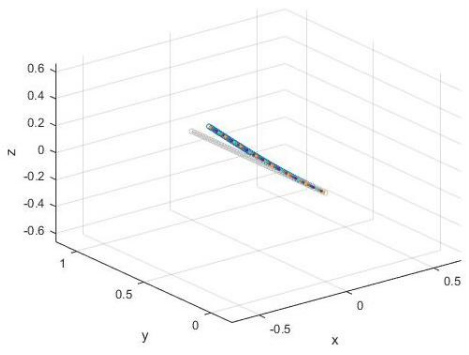

Figure 3: *Non-Tapered Beam in WFEM using 50 elements*

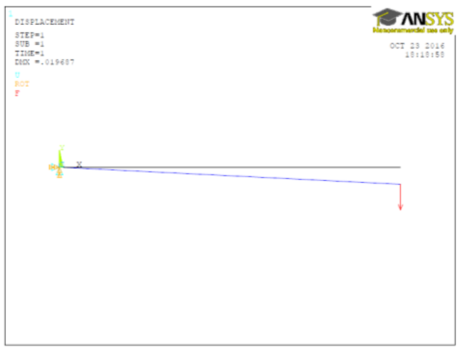

Figure 4: *Non-Tapered Beam in ANSYS using 1 element*

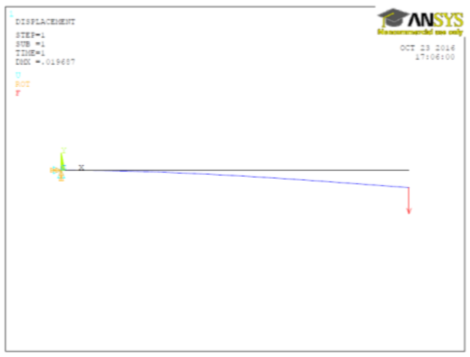

Figure 5: *Non-Tapered Beam in ANSYS using 50 elements*

Table 1: WFEM Analysis of a Non-Tapered Beam

Elements	**Displacement (m)**	**Frequency 1 (kHz)**	**Frequency 2 (kHz)**	**Frequency 3 (kHz)**	**Frequency 4 (kHz)**
1	0.0197	8.262	81.401	N/A	N/A
2	0.0197	8.227	51.968	175.766	510.148
5	0.0197	8.223	51.557	144.806	286.063
10	0.0197	8.223	51.533	144.325	283.017
25	0.0197	8.223	51.531	144.289	282.755
50	0.0197	8.223	51.531	144.288	282.748

Table 2: ANSYS Analysis of a Non-Tapered Beam

Elements	**Displacement (m)**
1	0.019687
2	0.019687
5	0.019687
10	0.019687
25	0.019687
50	0.019687

The data above, obtained from WFEM, shows that the accuracy of the deflection values do not increase as the element number increases. However, the mode shape frequencies seem to converge. The required number of elements for frequency convergence appears to increase for the higher order mode shapes. The calculated displacement from ANSYS closely matches that from WFEM. If the value from ANSYS is rounded to the same significant figure as WFEM than it would match exactly. ANSYS also followed the same pattern of not converging as the number of elements increase.

$$\delta = \frac{Fl^3}{3EI} \tag{1}$$

$$\delta = \frac{(10N)(1m)^3}{3(2.0408e10\ N/m^2)(.01m^4/12)} \tag{2}$$

$$\delta = 0.019599\ m \tag{3}$$

Solving the equations above results in a deflection value that is slightly less than ANSYS and WFEM. A finite element solution will never give the exact solution. However, it must determined whether the finite element solution is "good enough". In this case, the error between methods is only 0.45%, therefore the finite element solution is sufficiently accurate.

2. Rotated Non-Tapered Beam

It was desired to rotate a single element, in WFEM, to determine whether the transformation matrix accurately transforms the stiffness and matrix. The analysis is conducted on the same non-tapered beam shown in Figure 1 above, with the same boundary and loading conditions. Figure 6 below shows an un-rotated beam element deformed from a 10N load on the tip of the beam in the horizontal direction. The deflection of 0.0197m was previously determined in the above section.

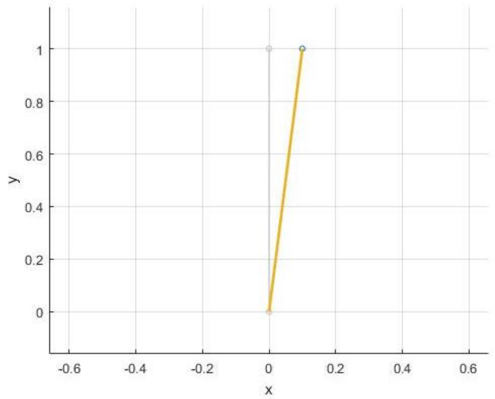

Figure 6: *Un-rotated, Non-Tapered Beam in WFEM using 1 element*

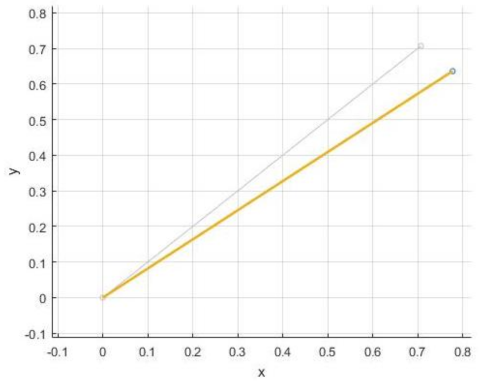

Figure 7: *Rotated, Non-Tapered Beam in WFEM using 1 element*

The same beam was rotated by a 45° rotation angle and deformed with a perpendicularly applied 10N load on the tip of the beam. This case is shown in Figure 7. The RSS method must be used to determine the deflection along the direction of the applied load. This is done by taking the square root of the sum of the squares of the x, y & z-components. Equations (4 & 5) show how the deflection is calculated. The calculated deflection matches that of the un-rotated case, therefore validating the accuracy of the transformation matrix.

$$D = \sqrt{x^2 + y^2 + z^2} \tag{4}$$

$$D = \sqrt{(0.0139)^2 + (0.0139)^2 + (0)^2} \tag{5}$$

$$D = 0.0197\ m \tag{6}$$

3. Tapered Beam

WFEM and ANSYS were also used to analyze a tapered, 2 noded, 3D beam element. Figure 8 below shows the tapered beam that was analyzed. All four sides of the square beam are tapered. The sides are reduced by 50% of their original length through the entire length of the beam, which is 1 meter. The boundary and loading conditions are shown as well as the geometry of the beam.

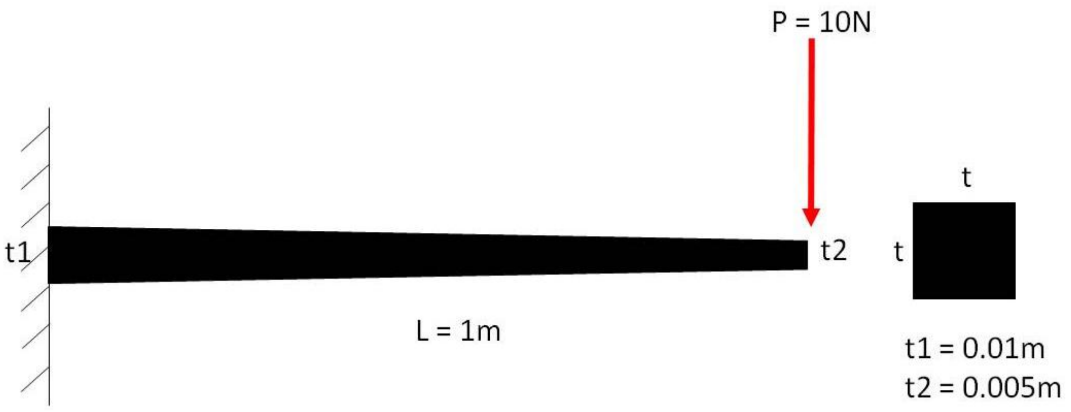

Figure 8: *Tapered Beam*

Mesh convergence studies were conducted in both WFEM and ANSYS. Figures 9 and 10 show the un-deformed and deformed tapered beam using WFEM with 1 element and 50 elements, respectively. Figures 11 and 12 show the un-deformed and deformed non-tapered beam using ANSYS with 1 element and 50 elements, respectively. Tables 3 and 4 below show the results obtained by WFEM and ANSYS , respectively. The natural frequencies for each mode were found using WFEM and are also shown in the Table 3.

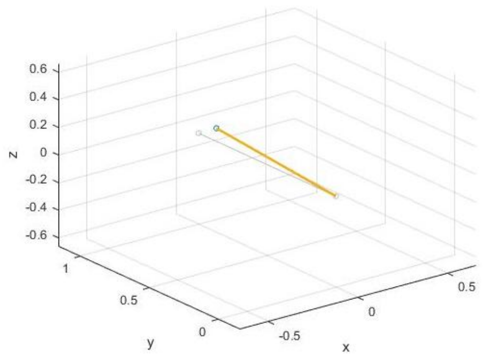

Figure 9: *Tapered Beam in WFEM using 1 element*

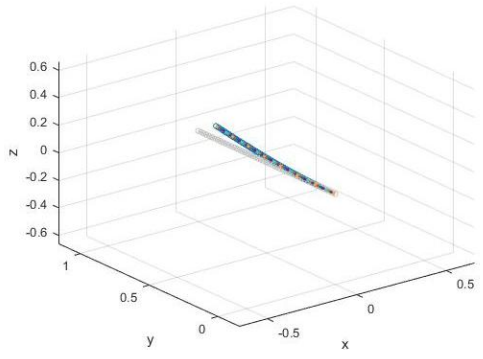

Figure 10: *Tapered Beam in WFEM using 50 elements*

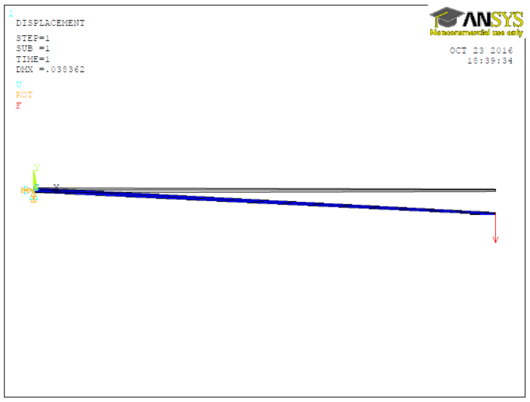
```
DISPLACEMENT
STEP=1
SUB =1
TIME=1
DMX =.038362
```

Figure 11: *Tapered Beam in ANSYS using 1 element*

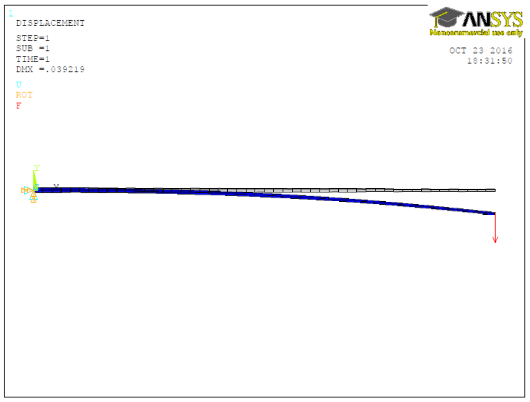
```
DISPLACEMENT
STEP=1
SUB =1
TIME=1
DMX =.039219
```

Figure 12: *Tapered Beam in ANSYS using 50 elements*

Table 3: WFEM Analysis of a Tapered Beam

Elements	**Displacement (m)**	**Frequency 1 (kHz)**	**Frequency 2 (kHz)**	**Frequency 3 (kHz)**	**Frequency 4 (kHz)**
1	0.0279	11.701	62.57	N/A	N/A
2	0.039	8.672	53.52	164.033	414.911
5	0.0456	7.244	45.769	129.379	256.305
10	0.0478	6.864	43.115	121.031	238.004
25	0.0491	6.655	41.725	116.88	229.16
50	0.0496	6.589	41.295	115.64	226.638

Table 4: ANSYS Analysis of a Tapered Beam

Elements	**Displacement (m)**
1	0.038362
2	0.039041
5	0.039212
10	0.039219
25	0.039219
50	0.039219

It can be seen from the data above that the calculated deflections converged to different values. The differences between values can explained through the calculation method. ANSYS calculates a tapered beam by stepping the cross-sectional area down throughout the length of the beam. This requires the calculation of constant cross-section elements. This method can get very close to the actual deflection but will never be exact. WFEM allows the user to linearly change the cross-section from node to node. This allows for a much more accurate result compared to ANSYS. Even though WFEM requires more elements to show convergence, the value of accuracy, with respect to computational costs, is much greater than ANSYS or other similar commercial solvers that use the same tapered methods.

4. Complex Truss

A box truss was chosen for the analysis of the complex beam element configuration. Figure 13 shows an image of a box truss. Box truss's are commonly used for building the lighting structures seen on stages and in arenas.

The truss was modeled in both WFEM and ANSYS. All 4 nodes on each end of the truss were fixed in all degrees of freedom. A 100N load was applied to each of the two vertical nodes in the center of the truss. Figure 14 shows an isometric view of the box truss modeled using the WFEM code in Matlab. Figure 15 and 16 show an isometric and top view, respectively, of the box truss modeled in ANSYS. All the plots show both the deformed and un-deformed shape of the truss. The maximum deflections found from both methods are listed in Table 6 and discussed in the Conclusion section.

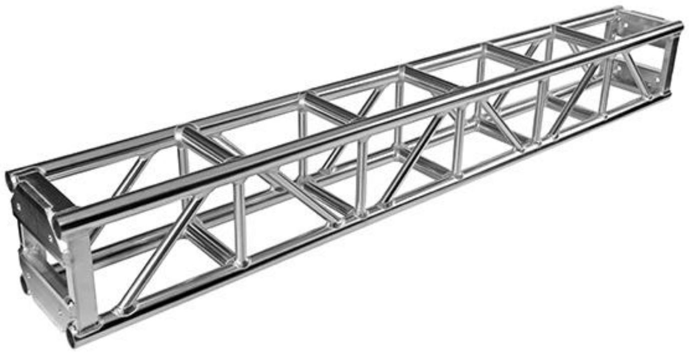

Figure 13: *Box Truss*

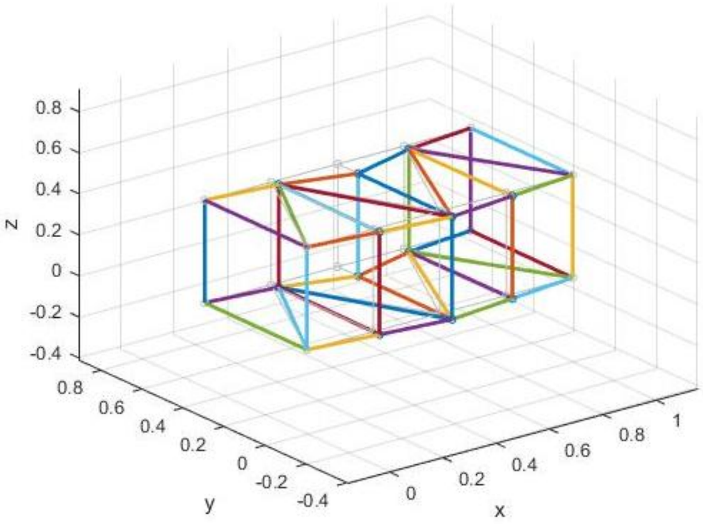

Figure 14: *Box Truss modeled in WFEM using beam elements*

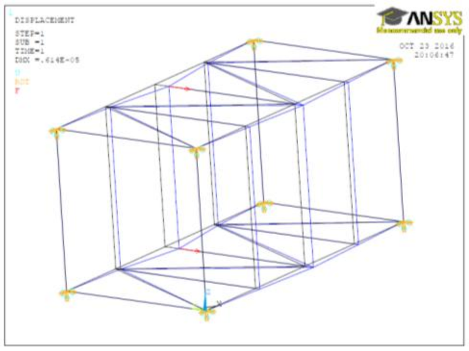

Figure 15: *Isometric view of Box Truss modeled in ANSYS using beam elements*

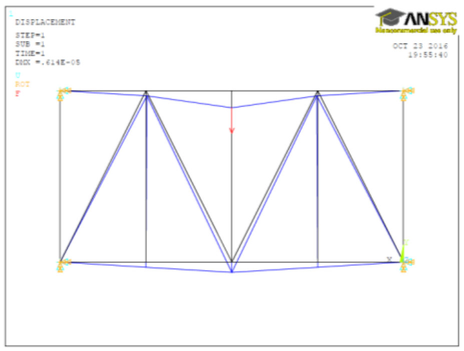

Figure 16: *Top view of Box Truss modeled in ANSYS using beam elements*

IV. Conclusion

The final results for Project 1 are shown in Table 6 below. Throughout this analysis WFEM has proven to be the best choice for a finite element solver using 3D beam elements. When analyzing a non-tapered beam, both methods yield similar results which are extremely close to the actual calculated value through the closed form solution. WFEM solves tapered beams much more accurate than ANSYS. The required computational power for both methods are similar and at an average level. However, versions of ANSYS can set you back a whopping $30,000 while WFEM currently has no cost. These reasons prove that WFEM is the better choice for the analyzed scenarios and chosen methods in this project.

Table 6: Results

	Non-Tapered Beam	Tapered Beam	Complex Truss
Method	**Displacement (m)**	**Displacement (m)**	**Displacement (m)**
WFEM	0.0197	0.0496	0.0372
ANSYS	0.0197	0.0392	6.14E-06
Closed Form	0.0196	N/A	N/A

V. Appendix

2 Noded beam element Matlab code:

```
function out=Project1(mode,b,c,d,e)

% BEAM3 does as listed below. It is an Euler-Bernoulli
% beam/rod/torsion model.
% Beam properties (bprops) are in the order
% bprops=[E G rho A1 A2 J1 J2 Ixx1 Ixx2 Iyy1 Iyy2]
% Fourth "node" defines the beam y plane and is actually from the
% points array.
%%
% Defining beam element properties in wfem input file:
% element properties
%   E G rho A1 A2 J1 J2 Izz1 Izz2 Iyy1 Iyy2
% Torsional rigidity, $J$, must be less than or equal
% to $Iyy+Izz$ at any given cross section.
%
% Defining beam2 element in wfem input file (Project1_Prompt):
%   node1 node2 pointnumber materialnumber

%
% See wfem.m for more explanation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Variables (global):
% -------------------
% K       :    Global stiffness matrix
% Ks      :    Global stiffness buckling matrix
% M       :    Global mass matrix
% nodes   :    [x y z] nodal locations
global ismatnewer
global K
global Ks
global M
global nodes % Node locations
global elprops
global element
global points
global Fepsn % Initial strain "forces".
global lines
global restart
global reload
global curlineno
global DoverL
global surfs
%
% Variables (local):
% ------------------
% bnodes  :    node/point numbers for actual beam nodes 1-2-3 and point
% k       :    stiffness matrix in local coordiates
% kg      :    stiffness matrix rotated into global coordinates
% m       :    mass matrix in local coordiates
% mg      :    mass matrix rotated into global coordinates
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Copyright Joseph C. Slater, 7/26/2002.
% joseph.slater@wright.edu
out=0;
if strcmp(mode,'numofnodes')
    % This allows a code to find out how many nodes this element has
    out=2;
end
if strcmp(mode,'generate')
  elnum=c;%When this mode is called, the element number is the 3rd
          %argument.
          %The second argument (b) is the element
          %definition. For this element b is
          %node1 node2 point(for rotation) and material#

          %There have to be 5 numbers for this element's
          %definition (above)
  if length(b)==4
      element(elnum).nodes=b(1:2);
      element(elnum).properties=b(4);
      element(elnum).point=b(3);
  else
      b
      %There have to be four numbers on a line defining the
      %element.
      warndlg(['Element ' num2str(elnum) ' on line ' ...
               num2str(element(elnum).lineno) ' entered incorrectly.'], ...
              ['Malformed Element'],'modal')
      return
  end

end

% Here we figure out what the beam properties mean. If you need
% them in a mode, that mode should be in the if on the next line.
if strcmp(mode,'make')||strcmp(mode,'istrainforces')
  elnum=b;% When this mode is called, the element number is given
          % as the second input.
  bnodes=[element(elnum).nodes element(elnum).point];% The point is
                                                     % referred to
                                                     % as node 4
                                                     % below,
                                                     % although it
                                                     % actually
                                                     % calls the
                                                     % array points
                                                     % to get its
                                                     % location. Its
                                                     % not really a
                                                     % node, but
                                                     % just a point
                                                     % that helps
                                                     % define
                                                     % orientation. Your
                                                     % element may
                                                     % not need
                                                     % such a
                                                     % reference point.
  bprops=elprops(element(elnum).properties).a;% element(elnum).properties
                                              % stores the
                                              % properties number
                                              % of the current
                                              % elnum. elprops
                                              % contains this
                                              % data. This is
                                              % precisely the
                                              % material properties
                                              % line in an
                                              % array. You can pull
                                              % out any value you
                                              % need for your use.

%
  if length(bprops)==11
      E=bprops(1);
      G=bprops(2);
      rho=bprops(3);
      A1=bprops(4);
      A2=bprops(5);
      J1=bprops(6);
      J2=bprops(7);
      Izz1=bprops(8);
      Izz2=bprops(9);
      Iyy1=bprops(10);
      Iyy2=bprops(11);
  else
      warndlg(['The number of material properties set for ' ...
               'this element (' num2str(length(bprops)) ') isn''t ' ...
               'appropriate for a beam3 element. '      ...
               'Please refer to the manual.'],...
              'Bad element property definition.','modal');
  end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Beam properties (bprops) are in the order
% bprops=[E G rho A1 A2 J1 J2 Izz1 Izz2 Iyy1 Iyy2]
% For a linear beam they are
% bprops=[E G rho A1 A2 J1 J2 Izz1 Izz2 Iyy1 Iyy2]

if strcmp(mode,'make')

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %
  % Define beam node locations for easy later referencing
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  x1=nodes(bnodes(1),1);
  y1=nodes(bnodes(1),2);
  z1=nodes(bnodes(1),3);
  x2=nodes(bnodes(2),1);
  y2=nodes(bnodes(2),2);
  z2=nodes(bnodes(2),3);
  x3=points(bnodes(3),1);
  y3=points(bnodes(3),2);
  z3=points(bnodes(3),3);

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %
  % Shape functions for higher order beam.
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Shape functions in matrix polynomial form (polyval style) for bending
  bn1 =  [  0.25   0   -0.75   0.5];
  bn1d =  [0.75 0 -0.75];
  bn1dd =  [1.5 0];
  bn2 =  [ 0.25 -0.25 -0.25 0.25];
  bn2d =  [0.75 -0.5 -0.25];
  bn2dd =  [1.5 -.5];
  bn3 =  [-0.25 0 0.75 0.5];
  bn3d = [-0.75 0 0.75];
  bn3dd =[-1.5 0];
  bn4 =  [0.25 0.25 -0.25 -0.25];
  bn4d = [0.75 0.5 -0.25];
  bn4dd =[1.5 0.5];

  % Shape functions in matrix polynomial form (polyval style) for
  % torsion/rod
  rn1=[-0.5 0.5];
  rn1d=[-0.5];
  rn2=[.5 .5];
  rn2d=[0.5];

  numbeamgauss=3; % Number of Gauss points for integration of beam element
  [bgpts,bgpw]=gauss(numbeamgauss);
  kb1=zeros(4,4);% For this beam, 2 nodes, 2DOF each, is a 4 by 4
                 % matrix.
  kb2=kb1; %Stiffness matrix for the x-z plane beam element.
  l=norm([x2 y2 z2]-[x1 y1 z1]);
  propertynum=num2str(element(elnum).properties);
  % Allowable aspect ratio. I recommend D/l=.1
  if isempty(DoverL)==1
    DoverL=.1;
  end
  %Euler bernoulli beams must be slender. Warn if not.
  if sqrt(A1*4/pi)/l>DoverL|sqrt(A2*4/pi)/l>DoverL
    warndlg({['Dimensions of element ' num2str(elnum) ' using properties '...
          propertynum ' are more suitable for a Timoshenko beam.'];...
         'radius divided by length is too large'},...
        'Improper application of element.','replace')
  end
  % This took some work, but provide bounds on other values.
  if (Izz1+Iyy1)<(1/2.1*A1^2/pi)|(Izz2+Iyy2)<(1/2.1*A2^2/pi)
    %2.0 would be exact for a circle
    warndlg({['Iyy+Izz for properties number' propertynum ' can''t be as '...
          'low as have been given.'];...
         'Nonphysical properties.'},['Impossible cross sectional' ...
            ' properties'],'replace')
  end
  slenderness=min([sqrt((Izz1+Iyy1)/A1) sqrt((Izz2+Iyy2)/A2)])/l;
  % Check if this is a beam or something so thin that its really a
  % string.
  if slenderness<.002
    disp([num2str(elnum) ['is a rediculously thin element. Please' ...
            ' check numbers.']])
  end

  Jac=l/2;% Beam Jacobian. valid only if node three is in the
          % middle of the beam. Luck for us, it always is (or the
          % code yells at you)
          % Local Bending in x-y plane
  for i=1:numbeamgauss
        beamsfs=[polyval(bn1dd,bgpts(i))/Jac^2;%evaluating second
                                           %derivatives of shape
                                           %functions to use in
                                           %generating stiffness
                                           %matrix. (at gauss point)
             polyval(bn2dd,bgpts(i))/Jac;
             polyval(bn3dd,bgpts(i))/Jac^2;
             polyval(bn4dd,bgpts(i))/Jac];

    Izz=polyval(rn1*Izz1+rn2*Izz2,bgpts(i));%Find Izz at
                                            %Gauss point
    kb1=kb1+bgpw(i)*beamsfs*beamsfs'*Izz*E*Jac;% This is the gauss
                                               % integration part
  end

% Local Bending in x-z plane
  for i=1:numbeamgauss
    beamsfs=[polyval(bn1dd,bgpts(i))/Jac^2;
             -polyval(bn2dd,bgpts(i))/Jac;
             polyval(bn3dd,bgpts(i))/Jac^2;
             -polyval(bn4dd,bgpts(i))/Jac];
    Iyy=polyval(rn1*Iyy1+rn2*Iyy2,bgpts(i));
    kb2=kb2+bgpw(i)*beamsfs*beamsfs'*Iyy*E*Jac;
  end

  % Local Extension in x, torsion about x
  numrodgauss=2;% Number of points to use for gauss point integration
  [rgpts,rgpw]=gauss(numrodgauss);
  krod=zeros(2,2);
  ktor=zeros(2,2);
  for i=1:numrodgauss
    rodsfs=[polyval(rn1d,rgpts(i))/Jac;
            polyval(rn2d,rgpts(i))/Jac];
    if (J1>(Iyy1+Izz1))|(J2>(Iyy2+Izz2))
      if (J1>(Iyy1+Izz1))
    disp('WARNING: J1 must be <= Iyy1+Izz1')%More checks for reality
      end
      if (J2>(Iyy2+Izz2))
    disp('WARNING: J2 must be <= Iyy2+Izz2')%More checks for reality
      end
        disp(['Error in element properties number '...
        num2str(element(elnum).properties) ...
        'used by element ' num2str(elnum) ' on line'...
        num2str(element(elnum).lineno) '.'])
    end
    J=polyval(rn1*J1+rn2*J2,rgpts(i));% J at gauss point.
    A=polyval(rn1*A1+rn2*A2,rgpts(i));% A at gauss point
    krod=krod+rgpw(i)*rodsfs*rodsfs'*A*E*Jac;%Since the shape
                                             %functions and Gauss
                                             %points are the same,
                                             %we are doing the rod
                                             %and torsion rod
                                             %together.
    ktor=ktor+rgpw(i)*rodsfs*rodsfs'*J*G*Jac;
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %
  % Derivation of Mass matrices
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  numbeamgauss=numbeamgauss+3; %Need more gauss points for the mass
                               %matrix.
  [bgpts,bgpw]=gauss(numbeamgauss);
  mb1=zeros(4,4); %initialize empty mass matrix
  % Local Bending in x-y plane
  for i=1:numbeamgauss
    beamsfs=[polyval(bn1,bgpts(i));
             polyval(bn2,bgpts(i))*Jac;
             polyval(bn3,bgpts(i));
             polyval(bn4,bgpts(i))*Jac];
    A=polyval(rn1*A1+rn2*A2,bgpts(i));
    mb1=mb1+bgpw(i)*beamsfs*beamsfs'*rho*A*Jac;%pause, and reflect
                                               %(OK, this was for debugging)
  end

  % Local Bending in x-z plane
  mb2=zeros(4,4);
  for i=1:numbeamgauss
    beamsfs=[polyval(bn1,bgpts(i));
             -polyval(bn2,bgpts(i))*Jac;
             polyval(bn3,bgpts(i));
             -polyval(bn4,bgpts(i))*Jac];
    A=polyval(rn1*A1+rn2*A2,bgpts(i));
    mb2=mb2+bgpw(i)*beamsfs*beamsfs'*rho*A*Jac;
  end

  % Local Extension in x, torsion about x
  numrodgauss=numrodgauss+1; %Need more gauss points for the mass
                             %matrix.
  [rgpts,rgpw]=gauss(numrodgauss);
  mrod=zeros(2,2); %initialize empty mass matrix
  mtor=zeros(2,2);
  for i=1:numrodgauss
    rodsfs=[polyval(rn1,rgpts(i));
            polyval(rn2,rgpts(i))];
    J=polyval(rn1*(Iyy1+Izz1)+rn2*(Iyy2+Izz2),rgpts(i));
    A=polyval(rn1*A1+rn2*A2,rgpts(i));
    mrod=mrod+rgpw(i)*rodsfs*rodsfs'*A*rho*Jac;
    mtor=mtor+rgpw(i)*rodsfs*rodsfs'*J*rho*Jac;
  end

  % Assembling each stiffness matrix into the complete elemental
  % stiffness matrix. We're just telling the sub-elements to be put
  % into the correct spots for the total element.
  k=zeros(12,12);
  k([2 6 8 12],[2 6 8 12])=kb1;
  k([3 5 9 11],[3 5 9 11])=kb2;
  k([1 7],[1 7])=krod;
```

```
  k([4 10],[4 10])=ktor;

  % Assembling each mass matrix into the complete elemental
  % mass matrix
  m=zeros(12,12);
  m([2 6 8 12],[2 6 8 12])=mb1;
  m([3 5 9 11],[3 5 9 11])=mb2;
  m([1 7],[1 7])=mrod;
  m([4 10],[4 10])=mtor;

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %
  % Coordinate rotations
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  R1=([x2 y2 z2]-[x1 y1 z1]);% Vector along element
  lam1=R1/norm(R1);% Unit direction
  R2=([x3 y3 z3]-[x1 y1 z1]);% Unit direction to point
  R2perp=R2-dot(R2,lam1)*lam1;% Part of R2 perpendicular to lam1
  udirec=0;
  while norm(R2perp)<10*eps% If R2perp is too small, (point in line
                           % with element, we need to cover the
                           % users a$$ and generate a point that
                           % isn't. We should put out a warning,
                           % but I commented it out.
    udirec=udirec+1;
    %disp('oops'); %This was my warning.
    %pause
    [minval,minloc]=min(lam1);
    R2perp=zeros(1,3);
    R2perp(udirec)=1;
    R2perp=R2perp-dot(R2perp,lam1)*lam1;
  end
  %Make the unit direction vectors for rotating and put them in the
  %rotation matrix.
  lam2=R2perp/norm(R2perp);
  lam3=cross(lam1,lam2);
  lamloc=[lam1;lam2;lam3];
  lam=sparse(12,12);
  lam(1:3,1:3)=lamloc;
  lam(4:6,4:6)=lamloc;
  lam(7:9,7:9)=lamloc;
  lam(10:12,10:12)=lamloc;

% $$$     lam=[lamloc z z z z z;
% $$$          z lamloc z z z z;
% $$$          z z lamloc z z z;
% $$$          z z z lamloc z z;
% $$$          z z z z lamloc z;
% $$$          z z z z z lamloc];
  element(elnum).lambda=lam;
  element(elnum).m=m;
  element(elnum).k=k;
%     eig(k)
%     pause
  kg=lam'*k*lam;
  mg=lam'*m*lam;
%   Trans = full(lam)

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %
  % Assembling matrices into global matrices
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  bn1=bnodes(1);bn2=bnodes(2);
  indices=[bn1*6+(-5:0) bn2*6+(-5:0)] ;


  K(indices,indices)=K(indices,indices)+kg;
  M(indices,indices)=M(indices,indices)+mg;


  % At this point we also know how to draw the element (what lines
  % and surfaces exist). For the beam3 element, 2 lines are
  % appropriate. Just add the pair of node numbers to the lines
  % array and that line will always be drawn.
  numlines=size(lines,1);
  lines(numlines+1,:)=[bn1 bn2];



  %If I have 4 nodes that I want to use to represent a surface, I
  %do the following.
  panelcolor=[1 0 1];% This picks a color. You can change the
                    % numbes between 0 and 1.
  %Don't like this color? Use colorui to pick another one. Another
  %option is that if we can't see the elements separately we can
  %chunk up x*y*z, divide by x*y*x of element, see if we get
  %integer powers or not to define colors that vary by panel.


  % You need to uncomment this line and assign values to node1,
  % node2, node3, and node4 in order to draw A SINGLE SURFACE. For
  % a brick, you need 6 lines like this.
  %surfs=[surfs;node1 node2 node3 node4 panelcolor];

  %Each surface can have a different color if you like. Just change
  %the last three numbers on the row corresponding to that
  %surface.

%diag(M)
elseif strcmp(mode,'istrainforces')
  % You don't need this
  % We need to have the stiffness matrix and the coordinate roation matrix.



elseif strcmp(mode,'draw')
elseif strcmp(mode,'buckle')
end
```

Single Element Non-Tapered Input file:

```
variables
%All of these actions are not the most efficient for this problem.
t=0.01
l=1
Ixx1=1/12*t^4
Ixx2=1/12*(t)^4
Iyy1=Ixx1
Iyy2=Ixx2
J1=0.95*(Ixx1+Iyy1)
J2=0.95*(Ixx2+Iyy2)

element properties
% Beam format
% E G rho A1    A2    J1 J2 Ixx1 Ixx2 Iyy1 Iyy2
steel    t^2  (t)^2 J1 J2 Ixx1 Ixx2 Iyy1 Iyy2
%Note that these were defined above. I can use variables in my input file.
% I also used "steel" as a property. WFEM has some of these predefined.
% Run "units" and type "who" to see variables/values available inside your
% input file

Project1 elements
%node1 node2 pointnum (beam properties number)
```

```
1 2 1 1

nodes
% I can include comment lines
% node num, x y z, Node number isn't ever stored in nodes matrix
1 0 0 0
2 0 1 0

points
1 1 1 1

fix clamp
1
% The preceeding put a clamp boundary condition on node 1.

%Ensure change correlates with Project1.m file
load
%node# dof# load
2 1 10

%fix pin
%1 0 0 1
%2 0 0 1
%
%fix surfaceball
%3 0 0 1
%
% The preceeding from fix pin on would create a simply supported
% beam in the x-y plane. Note that we needed to fix the translation
% in the z direction of the center node. We will still have torsion
% of the rod in it's middle about the x-axis, and extension of node
% 3 in the x-direction. Don't forget that the blank lines must be
% uncommented out to signify the end of data entry.

actions
modalanalysis
who
fs %dump sorted natural frequencies to the screen
% The stuff inside actions is simply executed at the wfem prompt. Any
% Matlab command can also be executed here. For example, you could double
% the mass matrix then do another modal analysis.
% This will display natural frequencies and prompt for mode shape display
%if uncommented
%modalreview
fsold=fs %Let's store them for later comparison
M=M/4; %Dividing M by 4 should double the natural frequencies
fs=[]; % WFEM won't run another modal analysis unless I force it to
%It's smart enough to know that it has already been done, so I need to
%clear the results to it is forced to regenerate them with another
%modalanalysis.
modalanalysis
disp('Natural Frequencies in KHz')
fprintf('%7.3f %7.3f\n',[fsold'; fs'])
disp('See, the natural frequency doubled as expected') % I wrote this out
%to the output for the user to read after execution.
% Uncommenting this line will cause an exit from WFEM immediately after
% execution
% end
%Let's do static analysis
staticanalysis
plotdeformed
%Here are the displacements, X
X
```

20 Element Non-Tapered Input file:

```
variables
%All of these actions are not the most efficient for this problem.
t=0.01
l=1
Ixx1=1/12*t^4
Ixx2=1/12*(t)^4
Iyy1=Ixx1
Iyy2=Ixx2
J1=0.95*(Ixx1+Iyy1)
J2=0.95*(Ixx2+Iyy2)

element properties
% Beam format
% E G rho A1    A2    J1 J2 Ixx1 Ixx2 Iyy1 Iyy2
steel    t^2  (t)^2 J1 J2 Ixx1 Ixx2 Iyy1 Iyy2
%Note that these were defined above. I can use variables in my input file.
% I also used "steel" as a property. WFEM has some of these predefined.
% Run "units" and type "who" to see variables/values available inside your
% input file

Project1 elements
%node1 node2 pointnum (beam properties number)
1 2 1 1
2 3 1 1
3 4 1 1
4 5 1 1
5 6 1 1
6 7 1 1
7 8 1 1
8 9 1 1
9 10 1 1
10 11 1 1
11 12 1 1
12 13 1 1
13 14 1 1
14 15 1 1
15 16 1 1
16 17 1 1
17 18 1 1
18 19 1 1
19 20 1 1
20 21 1 1


nodes
% I can include comment lines
% node num, x y z, Node number isn't ever stored in nodes matrix
1 0 0 0
2 0 l/20 0
3 0 2*l/20 0
4 0 3*l/20 0
5 0 4*l/20 0
6 0 5*l/20 0
7 0 6*l/20 0
8 0 7*l/20 0
9 0 8*l/20 0
10 0 9*l/20 0
11 0 10*l/20 0
12 0 11*l/20 0
13 0 12*l/20 0
```

```
14 0 13*l/20 0
15 0 14*l/20 0
16 0 15*l/20 0
17 0 16*l/20 0
18 0 17*l/20 0
19 0 18*l/20 0
20 0 19*l/20 0
21 0 20*l/20 0

points
1 1 1 1

fix clamp
1
% The preceeding put a clamp boundary condition on node 1.

%Ensure change correlates with Project1.m file
load
%node# dof# load
21 1 10

%fix pin
%1 0 0 1
%2 0 0 1
%
%fix surfaceball
%3 0 0 1
%
% The preceeding from fix pin on would create a simply supported
% beam in the x-y plane. Note that we needed to fix the translation
% in the z direction of the center node. We will still have torsion
% of the rod in it's middle about the x-axis, and extension of node
% 3 in the x-direction. Don't forget that the blank lines must be
% uncommented out to signify the end of data entry.
actions
modalanalysis
who
fs %dump sorted natural frequencies to the screen
% The stuff inside actions is simply executed at the wfem prompt. Any
% Matlab command can also be executed here. For example, you could double
% the mass matrix then do another modal analysis.
% This will display natural frequencies and prompt for mode shape display
%if uncommented
%modalreview
fsold=fs %Let's store them for later comparison
M=M/4; %Dividing M by 4 should double the natural frequencies
fs=[]; % WFEM won't run another modal analysis unless I force it to
%It's smart enough to know that it has already been done, so I need to
%clear the results to it is forced to regenerate them with another
%modalanalysis.
modalanalysis
disp('Natural Frequencies in KHz')
fprintf('%7.3f %7.3f\n',[fsold'; fs'])
disp('See, the natural frequency doubled as expected') % I wrote this out
%to the output for the user to read after execution.
% Uncommenting this line will cause an exit from WFEM immediately after
% execution
%end
%Let's do static analysis
staticanalysis
plotdeformed
%Here are the displacements, X
X
```

Complex Truss Input file:

```
variables
%All of these actions are not the most efficient for this problem.
t=0.01
l=1
Ixx1=1/12*t^4
Ixx2=1/12*(t)^4
Iyy1=Ixx1
Iyy2=Ixx2
J1=0.95*(Ixx1+Iyy1)
J2=0.95*(Ixx2+Iyy2)

element properties
% Beam format
% E G rho A1    A2    J1 J2 Ixx1 Ixx2 Iyy1 Iyy2
steel    t^2  (t)^2 J1 J2 Ixx1 Ixx2 Iyy1 Iyy2
%Note that these were defined above. I can use variables in my input file.
% I also used "steel" as a property. WFEM has some of these predefined.
% Run "units" and type "who" to see variables/values available inside your
% input file

Project1 elements
%node1 node2 pointnum (beam properties number)
1 2 1 1
2 3 1 1
3 4 1 1
4 5 1 1
5 6 1 1
6 7 1 1
7 8 1 1
8 9 1 1
9 10 1 1
10 1 1 1
1 9 1 1
9 2 1 1
9 3 1 1
3 8 1 1
3 7 1 1
7 4 1 1
7 5 1 1
1 11 1 1
2 12 1 1
3 13 1 1
4 14 1 1
5 15 1 1
6 16 1 1
7 17 1 1
8 18 1 1
9 19 1 1
10 20 1 1
11 12 1 1
12 13 1 1
13 14 1 1
14 15 1 1
15 16 1 1
16 17 1 1
17 18 1 1
18 19 1 1
19 20 1 1
20 11 1 1
11 19 1 1
```

```
19 12 1 1
19 13 1 1
13 18 1 1
13 17 1 1
17 14 1 1
17 15 1 1



nodes
% I can include comment lines
% node num, x y z, Node number isn't ever stored in nodes matrix
1 0 0 0
2 l/4 0 0
3 l/2 0 0
4 3*l/4 0 0
5 l 0 0
6 l l/2 0
7 3*l/4 l/2 0
8 l/2 l/2 0
9 l/4 l/2 0
10 0 l/2 0
11 0 0 l/2
12 l/4 0 l/2
13 l/2 0 l/2
14 3*l/4 0 l/2
15 l 0 l/2
16 l l/2 l/2
17 3*l/4 l/2 l/2
18 l/2 l/2 l/2
19 l/4 l/2 l/2
20 0 l/2 l/2




points
1 1 1 1

fix clamp
1
10
11
20
5
6
15
16

% The preceeding put a clamp boundary condition on node 1.

%Ensure change correlates with Project1.m file
load
%node# dof# load
8 2 -100
18 2 -100

%fix pin
%1 0 0 1
%2 0 0 1
%
%fix surfaceball
%3 0 0 1
```

```
%
% The preceeding from fix pin on would create a simply supported
% beam in the x-y plane. Note that we needed to fix the translation
% in the z direction of the center node. We will still have torsion
% of the rod in it's middle about the x-axis, and extension of node
% 3 in the x-direction. Don't forget that the blank lines must be
% uncommented out to signify the end of data entry.

actions
modalanalysis
who
fs %dump sorted natural frequencies to the screen
% The stuff inside actions is simply executed at the wfem prompt. Any
% Matlab command can also be executed here. For example, you could double
% the mass matrix then do another modal analysis.
% This will display natural frequencies and prompt for mode shape display
%if uncommented
%modalreview
fsold=fs %Let's store them for later comparison
M=M/4; %Dividing M by 4 should double the natural frequencies
fs=[]; % WFEM won't run another modal analysis unless I force it to
%It's smart enough to know that it has already been done, so I need to
%clear the results to it is forced to regenerate them with another
%modalanalysis.
modalanalysis
disp('Natural Frequencies in KHz')
fprintf('%7.3f %7.3f\n',[fsold'; fs'])
disp('See, the natural frequency doubled as expected') % I wrote this out
%to the output for the user to read after execution.
% Uncommenting this line will cause an exit from WFEM immediately after
% execution
% end
%Let's do static analysis
staticanalysis
plotdeformed
%Here are the displacements, X
X
```