**ME 7120 Finite Element Method Applications**

**Project 2**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Philip Bear, Chigozie Obidigbo, and Joel Summerfield

**Course:** FEMA

**Section:** ME-7120-01

**Prof:** Dr. Slater

­­­­­

**Submitted:** December 2nd, 2016

Table of Contents

[I. Nomenclature 2](#_Toc468285453)

[II. Table of Figures 2](#_Toc468285454)

[III. Project Description 1](#_Toc468285455)

[IV Brick Validation 1](#_Toc468285456)

[1. Extension Validation 2](#_Toc468285457)

[2. Shear Validation 4](#_Toc468285458)

[3. Patch Test 6](#_Toc468285459)

[V. Brick Application 9](#_Toc468285460)

[1. Tapered Beam 9](#_Toc468285461)

[2. Tapered Cylinder 11](#_Toc468285462)

[VI. Conclusion 13](#_Toc468285463)

[VII. Appendix 14](#_Toc468285464)

# Nomenclature

** = Displacement

P = Load

L = Length

A = Cross Sectional Area

E = Young’s Modulus of Elasticity

t = Thickness

# Table of Figures

[Figure 1: 3D Brick in Extension 2](#_Toc468285438)

[Figure 2: 2D Brick in Extension 3](#_Toc468285439)

[Figure 3: Force Applied to Elements 4](#_Toc468285440)

[Figure 4: Beam in Shear 5](#_Toc468285441)

[Figure 5: UnCorrected Brick in Shear 5](#_Toc468285442)

[Figure 6: Corrected Brick in Shear 6](#_Toc468285443)

[Figure 7: 2D Patch Test 7](#_Toc468285444)

[Figure 8: 3D Patch Test 8](#_Toc468285445)

[Figure 9: Tapered Beam – Brick 9](#_Toc468285446)

[Figure 10: Tapered Beam - Beam3 10](#_Toc468285447)

[Figure 11: Tapered Beam – ABAQUS 10](#_Toc468285448)

[Figure 12: Tapered Beam Displacement Results 11](#_Toc468285449)

[Figure 13: Tapered Cylinder - Brick 12](#_Toc468285450)

[Figure 14: Tapered Cylinder - ABAQUS 12](#_Toc468285451)

[Figure 15: Tapered Cylinder Displacement Results 13](#_Toc468285452)

# Project Description

The objective of this project is to produce a functioning brick element that can be used by WFEM for Finite Element Analysis. This brick must be validated in several ways that will be outlined within the contents of this report. In order to generate this functioning brick element, the following must be accomplished:

* Write a general code to generate the elemental stiffness matrix for a single brick element for WFEM
* Write a code to generate the elemental mass matrix
* Obtain the Finite Element matrices K in global coordinates
* Write a subroutine to assemble these elements into a global matrix
* Write a subroutine to apply specified boundary conditions
* Write a subroutine to solve for the entire displacement vector.

Once these tasks are completed and the brick element is validated it will be used to analyze a tapered beam and a tapered cylinder. These results will be validated and compared to results found through ABAQUS software.

# Brick Validation

### Extension Validation

The first validation the brick element had to pass was validating it would behave normally in extension. To do this validation, a constant load was applied at four corners of one face in the direction normal to the face. While this load was being applied, the opposite face of the brick was put under specific boundary conditions. One of the nodes clamped and therefore did not have freedom in any direction. Another node was pinned so that it could only move in one direction. The remaining two nodes were made surfaceballs and therefore could only move along a single plane. An image of the applied load to the brick can be seen in Figure 1.

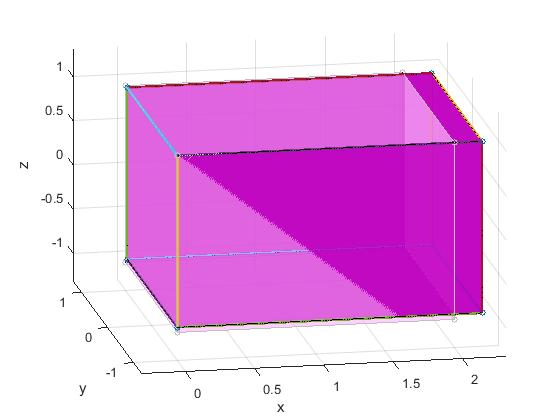


Figure : 3D Brick in Extension

As can be seen in Figure 1, The brick extends as expected. This can be seen even clearer in Figure 2, which is a 2D view of the brick in extension.

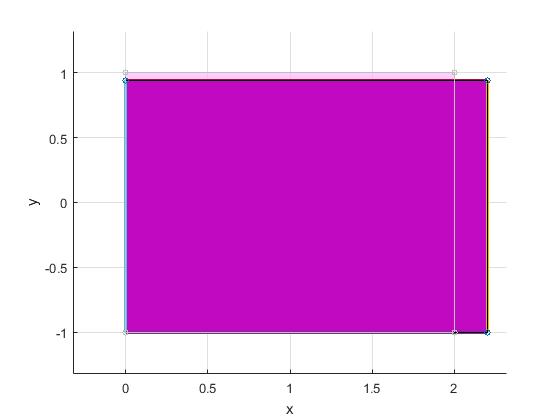


Figure : 2D Brick in Extension

In Figure 2, it can be seen that the brick deforms in two different directions. The pale lines are the original brick with no load applied, and the darker lines are the extended brick. With this in mind, it can be seen that the brick extends along the x axis and also contracts along the y axis. With the force being applied in the positive x direction, this makes sense. However, to finish the validation, the displacement calculated through WFEM has to be compared to the closed form solution. The closed form solution can be calculated through Equation (1).

(1)

To validate, a force of 2.5 Newtons was applied at each of the corners of the brick. The length of the brick was 2 meters. The cross sectional area of the brick was 4 square meters. The material for the brick was set to steel which resulted in a Young’s Modulus value of 203.4 GPa. The displacement found using both the closed form solution and the brick element was 2.46E-11 meters. For the brick element, this value was the displacement at all four corners of the face on which the force was applied. This process was repeated for all directions with the same result. Because the displacements match closed form, the brick element is validated.

### Shear Validation

The next validation necessary was validating the brick in shear. In order to do this, the brick was compared with the beam3example in WFEM. For both the brick and the beam a load was applied at the end of the element. The displacements of these elements were compared to validate the brick against the beam. The deflection of the beam was first compared to a brick without corrections for shear locking, and then to a corrected brick. An image of the force being applied to the beam and brick can be seen in Figure 3.

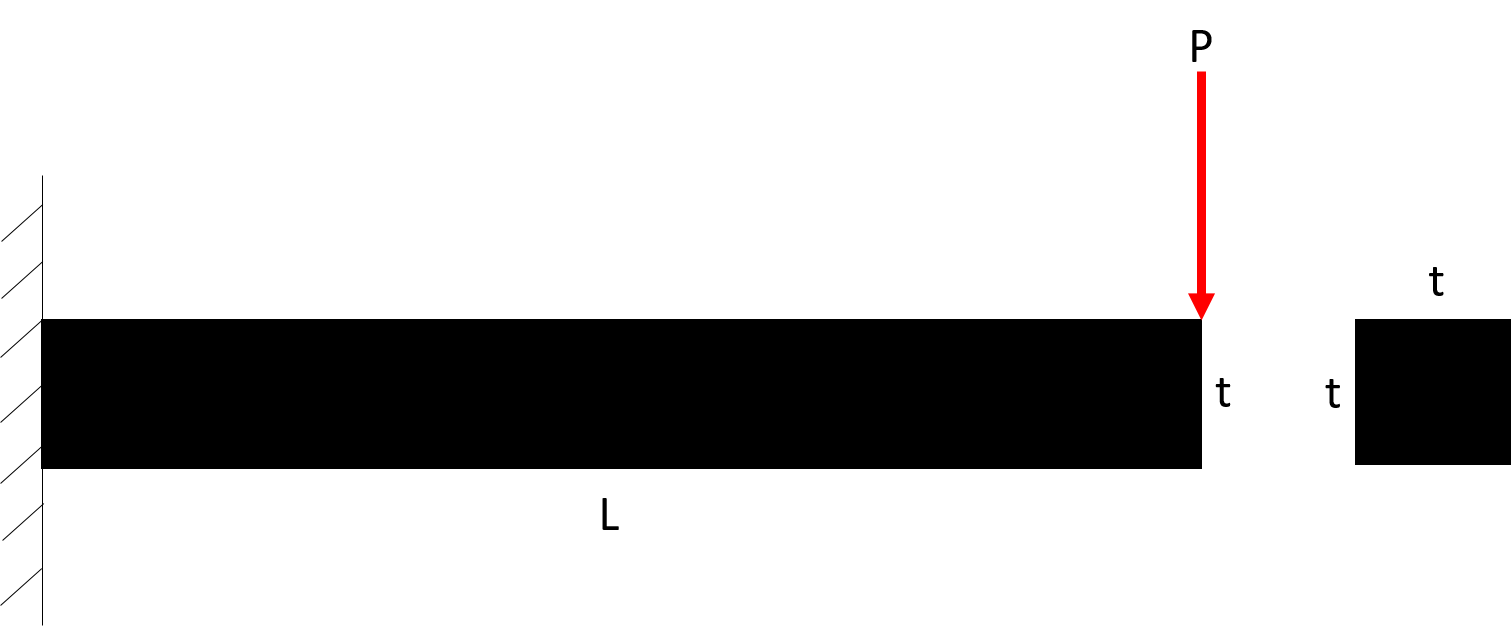


Figure : Force Applied to Elements

For this validation, the thickness of the element was set to 2 meters, the length was set to 5 meters. The load applied was 10 Newtons in the negative y direction. In Figure 4 the deflected beam element can be seen. Consequently, the deflection of the uncorrected brick and the corrected brick can be seen in Figure 5 and Figure 6, respectively.

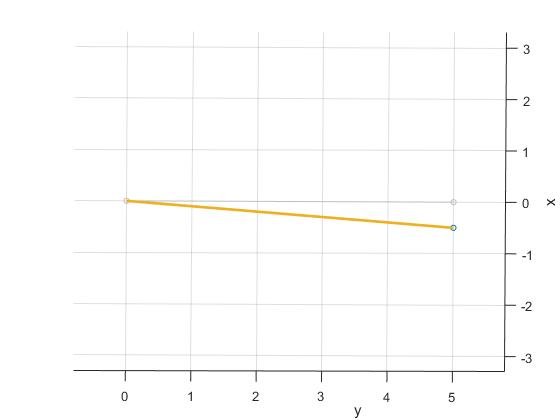


Figure : Beam in Shear

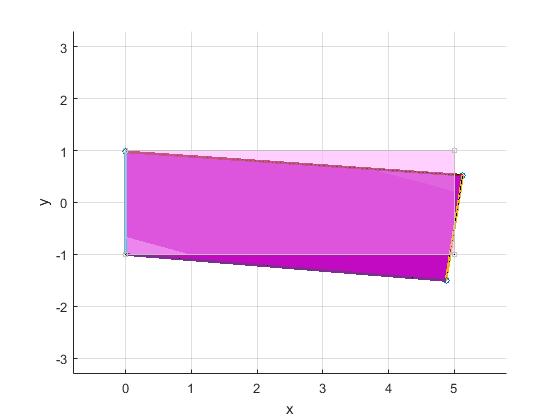


Figure : UnCorrected Brick in Shear

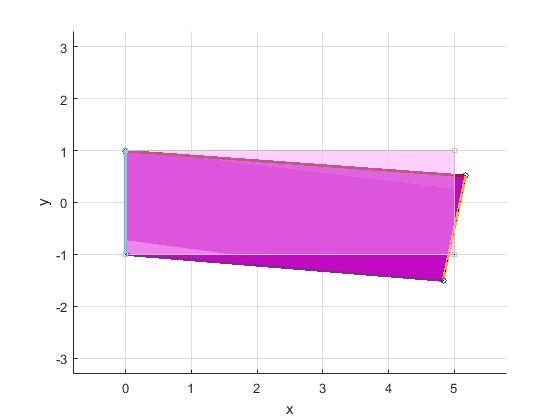


Figure : Corrected Brick in Shear

From the images, it is obvious that the bricks general behavior is correct, but this must also be validated by the deflection values. It is in the deflection values that the added corrections that deal with shear locking are seen. The deflection of the beam element was 1.54E-7 meters. The deflection of the uncorrected brick was 5.07E-8 meters. The deflection of the corrected brick was 1.34E-7 meters. While the values of the beam and corrected brick are not identical, they are close enough that it can be assumed the corrections are valid. The values will be difficult to match as the beam element is primarily for elements whose length is long in proportion to its thickness, while the brick works better for elements with uniform thickness and length. The shear test outlined above was then done in the other directions of shear to similar results, therefore the brick element was validated in shear.

### Patch Test

One of the last methods of validation for the brick element was the patch test. The patch test is where a single brick is broken into 8 elements, with the center node of the brick making each of the 8 elements non-uniform. Loading conditions which result in uniform strain are then applied and the strain at each of the corners of the original brick are compared. These values should be the same.

To apply the patch test, forces which result in uniform strain were applied to the corrected brick as prescribed. These forces were 100 Newtons at each of the corner nodes, 200 Newtons at the nodes bisecting each of the sides, and 400 Newtons at the center of the face. The visual result is shown in Figure 7 and Figure 8.

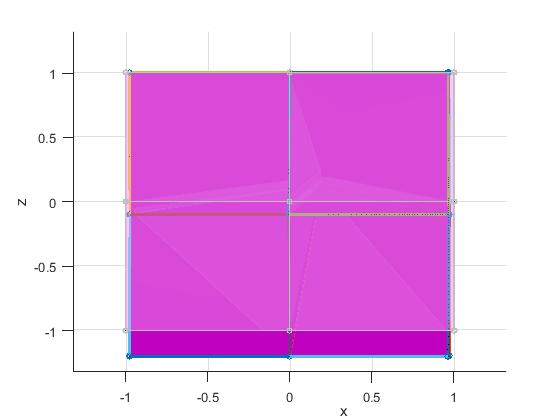


Figure : 2D Patch Test

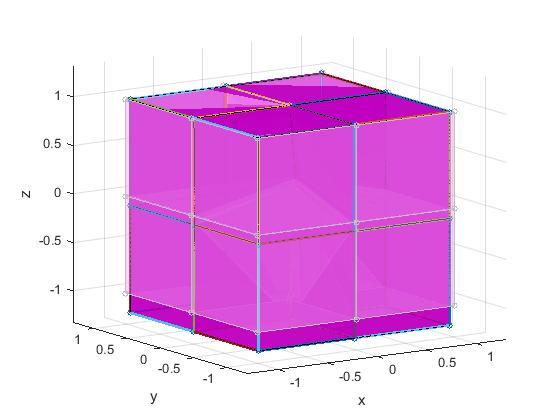


Figure : 3D Patch Test

When the loads stated above were applied to the brick of uniform 2 meter sides, the resulting displacement was 3.93E-9 meters. This displacement was constant at all nodes on the face on which the force was applied. To further validate, the result was compared to closed form using Equation 1. The closed form solution was found to be 3.93E-9 meters. Thus the brick element passed the patch test.

Note: The eigenvalues of the brick were also checked. As expected there were 6 zeros and all other values were positive.

# Brick Application

### Tapered Beam

Once the brick was validated it was used to calculate the tip displacement in two different structures. The first of these structures was a tapered beam. The displacement acquired by the brick through WFEM was found and then compared to the displacement acquired by ABAQUS. The results of both cases will also be compared to the displacement found by using the beam3 element in WFEM to add validity.

The image of the tapered beam by use of the brick element can be seen in Figure 9 and the tapered beam by use of ABAQUS can be seen in Figure 10. The tapered beam by use of the beam3 element can be seen in Figure 11.

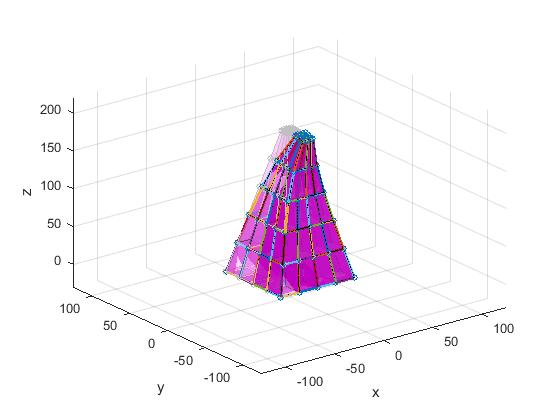


Figure : Tapered Beam – Brick

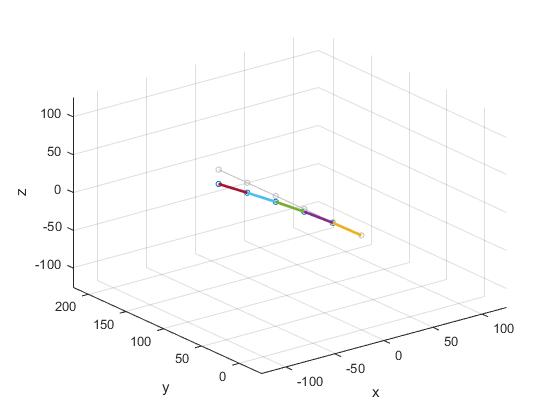


Figure : Tapered Beam - Beam3

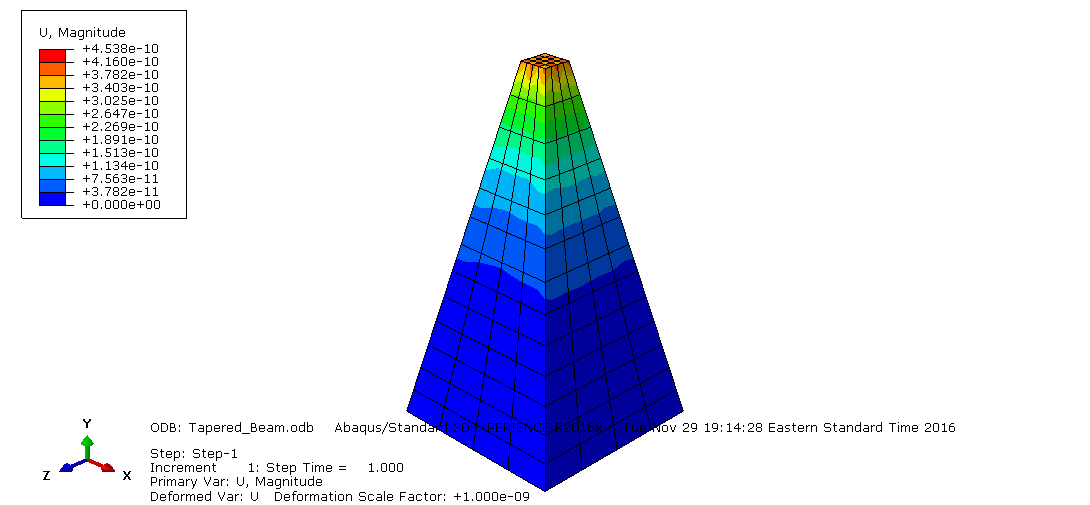


Figure : Tapered Beam – ABAQUS

The tapered beam has a height of 190 nanometers, a bottom side length of 74 nanometers, and a top side length of 10 nanometers. It is assumed that the top and bottom face along with the cross sectional area throughout has uniform side length, i.e. it is square. The load applied to the tip was 10 nanoNewtons.

When this tapered beam is evaluated through the three methods listed above, the displacements can be seen in Figure 12.



Figure : Tapered Beam Displacement Results

From Figure 12 it can be seen that the ABAQUS results predict a larger displacement than both the beam3 and brick element analysis. The main reason for this could possibly be that the ABAQUS software applies boundary conditions in a different way than the WFEM package. The reasoning behind this is because both the brick element and the beam3 element yielded similar results and both use WFEM. They both differ from ABAQUS so this would lead to the thought that there is something fundamentally different in the finite element packages. Application of boundary conditions is one of these things that could be fundamentally different.

### Tapered Cylinder

The second structure the brick element was used on was a tapered cylinder. As the beam3 element does not apply to this case, the displacement results of the brick element will only be compared to the displacement results found by ABAQUS. The visual representation of the displacement from the brick element can be seen in Figure 13 and the visual representation of the displacement from ABAQUS can be seen in Figure 14.

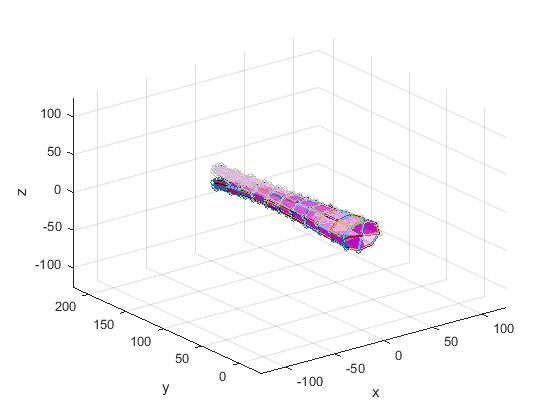


Figure : Tapered Cylinder - Brick

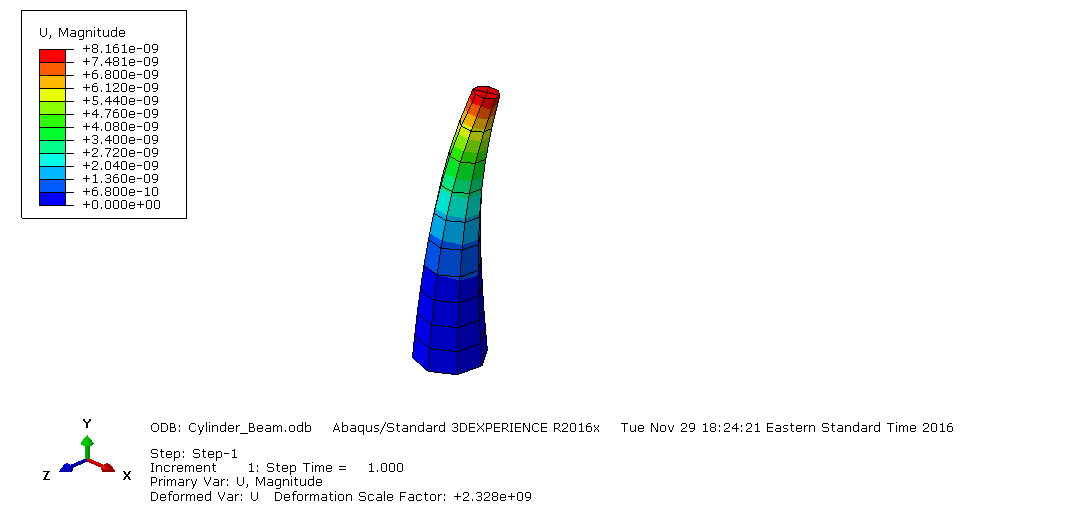


Figure : Tapered Cylinder - ABAQUS

The tapered cylinder has a height of 190 nanometers. Its bottom diameter was 74 nanometers and the top diameter is 10 nanometers. A load of 10 nanoNewtons was applied to the cylinder at the tip in the transverse direction.

The displacement results for the max displacement of the tapered cylinder can be seen in Figure 15.



Figure : Tapered Cylinder Displacement Results

As was seen with the primary tapered beam, the results between the two methods differ. For this case the max displacement found by ABAQUS is twice the max displacement found by the brick element. Once again, the most likely source of this error is the boundary conditions as described in the tapered beam section.

# Conclusion

Upon following the steps outlined in the project description a brick element was developed. This element was then validated in extension, shear, and through the use of the patch test. The extension validation was completed by applying an axial load and then checking the displacement results against the closed form solution. The shear validation was completed by applying a load that resulted in bending to the brick and comparing it to the already validated beam element. The displacement results of an uncorrected brick were compared to the beam and then in turn also compared to the results of a corrected brick. The corrected brick compared closely to the beam and the element was therefore validated in shear. The brick also passed the patch test by showing constant strain when broken into eight non-uniform smaller bricks and undergoing loads to produce constant strain in the element. By passing all of these validations, it was shown that the brick element was fully functional. It was then used to calculate displacement in a tapered beam and a tapered cylinder. The displacement results of these structures differed between the brick and ABAQUS, but this is mostly due to the difference in boundary conditions. In summary, a brick element was coded and validated and ultimately put into application for two different structures.

# Appendix

Brick

function out=Brick(mode,b,c,d,e)

% Brick does as listed below.

% Brick properties (bprops) are in the order

% bprops=[E G rho]

%%

% Defining beam element properties in wfem input file:

% element properties

% E G rho

%

% Defining Brick element in wfem input file:

% node1 node2 node3 node4 node5 node6 node7 node8 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 brick nodes 1-8

% 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

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

out=0;

if strcmp(mode,'numofnodes')

% This allows a code to find out how many nodes this element has

out=8;

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 node3 node4 node5 node6 node7 node8 materialnumber

%There have to be 9 elements for this element's

%definition (above)

if length(b)==9

element(elnum).nodes=b(1:8);

element(elnum).properties=b(9);

else

b

%There have to be nine 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];% 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)==3

E=bprops(1);

G=bprops(2);

rho=bprops(3);

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Brick properties (bprops) are in the order

% bprops=[E G rho]

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=nodes(bnodes(3),1);

y3=nodes(bnodes(3),2);

z3=nodes(bnodes(3),3);

x4=nodes(bnodes(4),1);

y4=nodes(bnodes(4),2);

z4=nodes(bnodes(4),3);

x5=nodes(bnodes(5),1);

y5=nodes(bnodes(5),2);

z5=nodes(bnodes(5),3);

x6=nodes(bnodes(6),1);

y6=nodes(bnodes(6),2);

z6=nodes(bnodes(6),3);

x7=nodes(bnodes(7),1);

y7=nodes(bnodes(7),2);

z7=nodes(bnodes(7),3);

x8=nodes(bnodes(8),1);

y8=nodes(bnodes(8),2);

z8=nodes(bnodes(8),3);

xvec=[x1 x2 x3 x4 x5 x6 x7 x8]; % stores nodal x values in J\_brick format

yvec=[y1 y2 y3 y4 y5 y6 y7 y8]; % stores nodal y values in J\_brick format

zvec=[z1 z2 z3 z4 z5 z6 z7 z8]; % stores nodal z values in J\_brick format

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%

% Define gauss points and loop to integrate for Ke and Me

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Emat=E\_matrix(E,G); % Calculates the E\_matrix for the element material

Me=zeros(24,24); % Preallocates the element mass matrix

Ke=zeros(33,33); % Preallocates the element stiffness matrix

numbrickgauss=5; % Chooses a number of gauss points for the stiffness matrix

[bgpts,bgpw]=gauss([numbrickgauss,numbrickgauss,numbrickgauss]); % finds the gauss points and weights

J0=J\_brick(0,0,0,xvec,yvec,zvec);

for i=1:size(bgpts,1) % loops over all the gauss points

[J,dNdx,dNdy,dNdz]=J\_brick(bgpts(i,1),bgpts(i,2),bgpts(i,3),xvec,yvec,zvec); % Finds J for the current Gauss point

B=B\_brick(dNdx,dNdy,dNdz); % Finds B for the current Gauss point

Bt=B'; % Calculates B transpose

Ki=bgpw(i)\*Bt\*Emat\*B\*det(J); % Calculates the weighted Gauss point stiffness

Ke(1:24,1:24)=Ke(1:24,1:24)+Ki(1:24,1:24); % adds the weighted Gauss point stiffness to the element stiffness

Ba=Ba\_brick(J,bgpts(i,1),bgpts(i,2),bgpts(i,3));

B=[B Ba];

Bt=B'; % Calculates B transpose

Ki=bgpw(i)\*Bt\*Emat\*B\*det(J0); % Calculates the weighted Gauss point stiffness

Ke(1:33,25:33)=Ke(1:33,25:33)+Ki(1:33,25:33); % adds the weighted Gauss point stiffness to the element stiffness

Ke(25:33,1:24)=Ke(25:33,1:24)+Ki(25:33,1:24);

end

numbrickgauss=numbrickgauss+1; % Adds more gauss points for the mass matrix

[bgpts,bgpw]=gauss([numbrickgauss,numbrickgauss,numbrickgauss]); % finds the gauss points and weights

for i=1:size(bgpts,1) % loops over all the gauss points

J=J\_brick(bgpts(i,1),bgpts(i,2),bgpts(i,3),xvec,yvec,zvec); % Finds J for the current Gauss point

N=N\_brick(bgpts(i,1),bgpts(i,2),bgpts(i,3)); % Finds N for the current Gauss point

Nt=N'; % Calculates N transpose

Mi=bgpw(i)\*Nt\*rho\*N\*det(J); % Calculates the weighted Gauss point mass

Me=Me+Mi; % adds the weighted Gauss point mass to the element mass

end

[Ke]=ZeroDOFs(Ke);

[Ke,Me]=Guyan\_Brick(Ke,Me);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%

% Assembling matrices into global matrices

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

bn1=bnodes(1);

bn2=bnodes(2);

bn3=bnodes(3);

bn4=bnodes(4);

bn5=bnodes(5);

bn6=bnodes(6);

bn7=bnodes(7);

bn8=bnodes(8);

indices=[bn1\*6+(-5:-3) bn2\*6+(-5:-3) bn3\*6+(-5:-3) bn4\*6+(-5:-3)...

bn5\*6+(-5:-3) bn6\*6+(-5:-3) bn7\*6+(-5:-3) bn8\*6+(-5:-3)] ;

K(indices,indices)=K(indices,indices)+Ke;

M(indices,indices)=M(indices,indices)+Me;

% 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];

lines(numlines+2,:)=[bn2 bn3];

lines(numlines+3,:)=[bn3 bn4];

lines(numlines+4,:)=[bn4 bn1];

lines(numlines+5,:)=[bn5 bn6];

lines(numlines+6,:)=[bn6 bn7];

lines(numlines+7,:)=[bn7 bn8];

lines(numlines+8,:)=[bn8 bn5];

lines(numlines+9,:)=[bn1 bn5];

lines(numlines+10,:)=[bn2 bn6];

lines(numlines+11,:)=[bn3 bn7];

lines(numlines+12,:)=[bn4 bn8];

%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;bn1 bn2 bn3 bn4 panelcolor];

surfs=[surfs;bn2 bn6 bn7 bn3 panelcolor];

surfs=[surfs;bn6 bn5 bn8 bn7 panelcolor];

surfs=[surfs;bn5 bn1 bn4 bn8 panelcolor];

surfs=[surfs;bn4 bn8 bn7 bn3 panelcolor];

surfs=[surfs;bn1 bn5 bn6 bn2 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

E\_mat

function [ Emat ] = E\_matrix( E,G )

% E\_matrix solves for Emat given E and G

nu=(E/(2\*G))-1; % Poisson's Ratio

val1=1-nu; % Calculates values used in the matrix

val2=1+nu;

val3=1-2\*nu;

Emat=zeros(6,6); % Creates a matrix of all zeros

Emat(1:3,1:3)=nu; % Fills the top left 3x3 block with nu

Emat(1,1)=val1; % Overwrites the diagonal

Emat(2,2)=val1;

Emat(3,3)=val1;

Emat(4,4)=val3/2; % Writes the bottom right diagonal

Emat(5,5)=val3/2;

Emat(6,6)=val3/2;

Emat=Emat\*E/(val2\*val3); % Multiplies by the outside constants

end

Guyan\_Brick

function [ Kr,Mr ] = Guyan\_Brick( Ke,Me )

% Guyan\_Brick reduces the stiffness and mass matrix size to 24x24 using

% guyan reduction

% check to see if Ke needs to be reduced

if size(Ke,1)>24 || size(Ke,2)>24

K11=Ke(1:24,1:24);

K21=Ke(25:end,1:24);

K22=Ke(25:end,25:end);

T=[eye(size(K11,1));-K22\K21];

Kr=T'\*Ke\*T;

else

Kr=Ke;

end

% check to see if Me needs to be reduced

if size(Me,1)>24 || size(Me,2)>24

M11=Me(1:24,1:24);

M21=Ke(25:end,1:24);

M22=Me(25:end,25:end);

T=[eye(size(M11,1));-M22\M21];

Mr=T'\*Me\*T;

else

Mr=Me;

end

end

J\_Brick

function [ J,dNdx,dNdy,dNdz ] = J\_brick( Epsilon, Eta, Zeta, x, y, z )

% J\_brick takes given Epsilon, Eta, Zeta (doubles), x, y, and z (1x8 vectors)

% for a brick element and returns J. Epsilon, Eta, and Zeta go from -1 to 1.

% Epsilon is Xi in the notes. EpsilonI, EtaI, and ZetaI are the nodal

% values in order

EpsilonI=[-1 -1 -1 -1 1 1 1 1];

EtaI=[-1 -1 1 1 -1 -1 1 1];

ZetaI=[1 -1 -1 1 1 -1 -1 1];

for i=1:8 % iterates over each node

dNdEpsilon(i)=(1/8)\*EpsilonI(i)\*(1+Eta\*EtaI(i))\*(1+Zeta\*ZetaI(i));

dNdEta(i)=(1/8)\*EtaI(i)\*(1+Epsilon\*EpsilonI(i))\*(1+Zeta\*ZetaI(i));

dNdZeta(i)=(1/8)\*ZetaI(i)\*(1+Epsilon\*EpsilonI(i))\*(1+Eta\*EtaI(i));

end

J=[dNdEpsilon; dNdEta; dNdZeta]\*[x' y' z']; % assembles derivative and nodal locations matrices and multiplies

for i=1:8

Resultant=J\[dNdEpsilon(i); dNdEta(i); dNdZeta(i)]; % finds the cartesian derivatives

dNdx(i)=Resultant(1); % stores the current nodal x derivative

dNdy(i)=Resultant(2); % stores the current nodal y derivative

dNdz(i)=Resultant(3); % stores the current nodal z derivative

end

N\_Brick

function [ J,dNdx,dNdy,dNdz ] = J\_brick( Epsilon, Eta, Zeta, x, y, z )

% J\_brick takes given Epsilon, Eta, Zeta (doubles), x, y, and z (1x8 vectors)

% for a brick element and returns J. Epsilon, Eta, and Zeta go from -1 to 1.

% Epsilon is Xi in the notes. EpsilonI, EtaI, and ZetaI are the nodal

% values in order

EpsilonI=[-1 -1 -1 -1 1 1 1 1];

EtaI=[-1 -1 1 1 -1 -1 1 1];

ZetaI=[1 -1 -1 1 1 -1 -1 1];

for i=1:8 % iterates over each node

dNdEpsilon(i)=(1/8)\*EpsilonI(i)\*(1+Eta\*EtaI(i))\*(1+Zeta\*ZetaI(i));

dNdEta(i)=(1/8)\*EtaI(i)\*(1+Epsilon\*EpsilonI(i))\*(1+Zeta\*ZetaI(i));

dNdZeta(i)=(1/8)\*ZetaI(i)\*(1+Epsilon\*EpsilonI(i))\*(1+Eta\*EtaI(i));

end

J=[dNdEpsilon; dNdEta; dNdZeta]\*[x' y' z']; % assembles derivative and nodal locations matrices and multiplies

for i=1:8

Resultant=J\[dNdEpsilon(i); dNdEta(i); dNdZeta(i)]; % finds the cartesian derivatives

dNdx(i)=Resultant(1); % stores the current nodal x derivative

dNdy(i)=Resultant(2); % stores the current nodal y derivative

dNdz(i)=Resultant(3); % stores the current nodal z derivative

end