ME 7120 FEA Project 2

8 Node Brick Element Formulation and Verification

Admir Makas

7/12/16

**Introduction**

The goal of the project is to formulate the 8 node brick element, which belongs to the 3-D element family. In the formulation process the element shear locking relaxation was introduced by including 3 node-less degrees of freedom. This, along with the original 8 shape functions resulted in 11 shape functions total. In order to obtain a 24X24 stiffness matrix that is expected for this element type, static condensation was implemented, which cancelled out the redundant degrees of freedom yielding the final 24X24 stiffness matrix (K).

Element formulation was implemented within the WFEM framework and validated against Nastran FEA package. Two test cases used in this report are cantilevered tapered cylinder and pyramid that are subjected to a tip load. The project requirements supplied the dimensional data in nanometers. However, this unit was cumbersome to use in model constriction so the decision was made to instead model the cylinder and pyramid using meters.

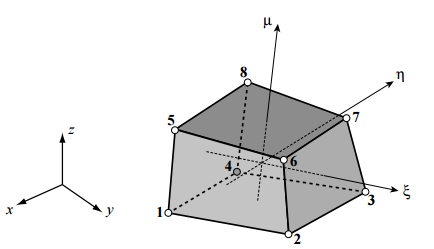
Additionally, results from the 3-D brick implementation were compared to 1-D beam element results from both the Nastran and WFEM solutions.

For all the verification runs, isotropic steel material properties were used with following values: E = 200 GPa, ν = 0.30, and ρ = 8000 .

Because of the size of the structure, a load of 1e11 Newtons was applied to the tip, which was able to generate a reasonable tip deflection that is within the linear regime of the material response.

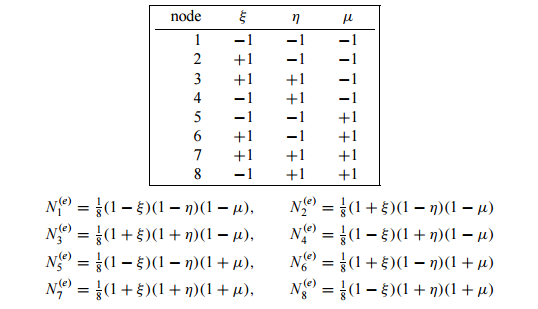
**Element Formulation:**

Typical brick element layout can be seen in Figure 1. Here we see node definitions along with global and local coordinates.



**Figure 1**: Eight node brick element

For this element, following node numbering and shape functions were defined in the local coordinate system.



**Figure 2**: Node definitions and shape functions.

In addition to the shape functions defined above, three additional shape functions are introduced for the node-less degrees of freedom. These additional degrees of freedom are needed in order to implement shear locking relaxation. Figure 3 shows these additional shape functions.

**Figure 3**: Shape functions for node-less degrees of freedom.

Information listed above was implemented in Matlab to construct the element. Code implementation can be seen below. Some code was omitted from the m-file due to brevity. However, all the pertinent implementation are present below. It should also be noted that the code enables the user to decide how many integration points are desired. Nonetheless, it was discovered that the stiffness matrix formulation is already converged using 2 integration points. Therefore, it is not necessary to requires higher number of integration points.

----ELEMENT CODE STARTS HERE

global\_nodes = [[x1 y1 z1]; [x2 y2 z2]; [x3 y3 z3]; [x4 y4 z4];...

[x5 y5 z5]; [x6 y6 z6]; [x7 y7 z7]; [x8 y8 z8]];

% Number of Gauss points for integration of BRICK element

% numbeamgauss=2; [bgpts,bgpw]=gauss(numbeamgauss);

num\_gauss=2

[int\_p, int\_w] = gauss(num\_gauss)

intPts=zeros(num\_gauss^3,3);

intWts=zeros(num\_gauss^3,3);

index=0;

for i=1:num\_gauss

for j=1:num\_gauss

for k=1:num\_gauss

index=index+1

intPts(index,:) = [int\_p(i) int\_p(j) int\_p(k)];

intWts(index,:) = [int\_w(i) int\_w(j) int\_w(k)];

end

end

end

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

%

% Derivation of Stiffness and Mass matrix

%

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

Bd=zeros(6,24);

Ba=zeros(6,9);

Me=zeros(24,24);

Kbb = zeros(24,24);

Kba = zeros(24,9);

Kab = zeros(9,24);

Kaa = zeros(9,9);

Em=getE(E,mu);

%==========================================================================

%Need dN evaluated at location r=s=t=0 for the incompatible modes

dN0=getdN(0,0,0);

%Get Jacobian evaluated at the center of the brick element used for

%Gauss integration of incompatible modes

J0=dN0\*global\_nodes;

Jinv0=J0\eye(3);

%==========================================================================

% Loop to construct BRICK stiffness matrix

for p=1:num\_gauss^3

r = intPts(p,1);

s = intPts(p,2);

t = intPts(p,3);

Ne=getN(r,s,t);

dN=getdN(r,s,t);

%Derivatives of incompatible shape functions

dNa=getdNa(r,s,t);

J=dN\*global\_nodes;

Jinv=J\eye(3);

JDet = det(J);

for q=1:11

if q<=8

dN\_i = dN(:,q);

Bi=[Jinv(1,:)\*dN\_i 0 0;

0 Jinv(2,:)\*dN\_i 0;

0 0 Jinv(3,:)\*dN\_i;

Jinv(2,:)\*dN\_i Jinv(1,:)\*dN\_i 0;

0 Jinv(3,:)\*dN\_i Jinv(2,:)\*dN\_i;

Jinv(3,:)\*dN\_i 0 Jinv(1,:)\*dN\_i];

Bd(1:end, 1+(q-1)\*3:1+(q-1)\*3+2) = Bi(1:end, 1:end);

else

dN\_i = dNa(:,q-8);

Bi=[Jinv0(1,:)\*dN\_i 0 0;

0 Jinv0(2,:)\*dN\_i 0;

0 0 Jinv0(3,:)\*dN\_i;

Jinv0(2,:)\*dN\_i Jinv0(1,:)\*dN\_i 0;

0 Jinv0(3,:)\*dN\_i Jinv0(2,:)\*dN\_i;

Jinv0(3,:)\*dN\_i 0 Jinv0(1,:)\*dN\_i];

Ba(1:end, 1+(q-1-8)\*3:1+(q-1-8)\*3+2) = Bi(1:end, 1:end);

end

end

Kbbi = prod(intWts(p,1:end))\*JDet\*(Bd'\*Em\*Bd);

Kbai = prod(intWts(p,1:end))\*JDet\*(Bd'\*Em\*Ba);

Kabi = prod(intWts(p,1:end))\*JDet\*(Ba'\*Em\*Bd);

Kaai = prod(intWts(p,1:end))\*JDet\*(Ba'\*Em\*Ba);

Kbb = Kbb+Kbbi;

Kba = Kba+Kbai;

Kab = Kab+Kabi;

Kaa = Kaa+Kaai;

Mi = rho\*prod(intWts(p,1:end))\*JDet\*(Ne'\*Ne);

Me=Me+Mi;

end

Ke = Kbb - Kba\*(Kaa\Kab); **%Static condensation**

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

%

% Assembling matrices into global matrices

%

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

element(elnum).m=Me;

element(elnum).k=Ke;

indices = zeros(1,24);

for w = 1:8

node\_i = element(elnum).nodes(w);

indices(3\*w-2:3\*w) = 1+(node\_i-1)\*6:3+(node\_i-1)\*6;

end

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

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

NodeId = reshape(element(elnum).nodes, [4,2])';

NodeId = [NodeId NodeId(:,1)];

% % Connecting node information

% 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; NodeId(1,1:4) panelcolor];

surfs=[surfs; NodeId(2,1:4) panelcolor];

for nn = 1:4

surfs=[surfs; NodeId(1,nn:nn+1) NodeId(2,nn+1) ...

NodeId(2,nn) panelcolor];

end

%Each surface can have a different color if you like. Just change

%the last three numbers on the row corresponding to that

%surface.

end

surface\_maker;

end

----ELEMENT CODE END HERE

Code above uses additional functions that help to define the **N**, **B**, and **E** matrix. Their respective code is listed below:

Function defines Ne matrix for shape functions 1-8:

function [Ne] = getN(r, s, t)

N1 = 1/8\*((1-r)\*(1-s)\*(1-t));

N2 = 1/8\*((1+r)\*(1-s)\*(1-t));

N3 = 1/8\*((1+r)\*(1+s)\*(1-t));

N4 = 1/8\*((1-r)\*(1+s)\*(1-t));

N5 = 1/8\*((1-r)\*(1-s)\*(1+t));

N6 = 1/8\*((1+r)\*(1-s)\*(1+t));

N7 = 1/8\*((1+r)\*(1+s)\*(1+t));

N8 = 1/8\*((1-r)\*(1+s)\*(1+t));

Ne=[N1\*eye(3) N2\*eye(3) N3\*eye(3) N4\*eye(3) N5\*eye(3) N6\*eye(3)...

N7\*eye(3) N8\*eye(3)];

end

Function defines derivatives of the shape functions used to generate **B**:

function [dN] = getdN(r, s, t) %Computes dN for stiffness matrix calc.

dN1r = -((s - 1)\*(t - 1))/8;

dN1s = -((r - 1)\*(t - 1))/8;

dN1t = -((r - 1)\*(s - 1))/8;

dN2r = ((s - 1)\*(t - 1))/8;

dN2s = ((r + 1)\*(t - 1))/8;

dN2t = ((r + 1)\*(s - 1))/8;

dN3r = -((s + 1)\*(t - 1))/8;

dN3s = -((r + 1)\*(t - 1))/8;

dN3t = -((r + 1)\*(s + 1))/8;

dN4r = ((s + 1)\*(t - 1))/8;

dN4s = ((r - 1)\*(t - 1))/8;

dN4t = ((r - 1)\*(s + 1))/8;

dN5r = ((s - 1)\*(t + 1))/8;

dN5s = ((r - 1)\*(t + 1))/8;

dN5t = ((r - 1)\*(s - 1))/8;

dN6r = -((s - 1)\*(t + 1))/8;

dN6s = -((r + 1)\*(t + 1))/8;

dN6t = -((r + 1)\*(s - 1))/8;

dN7r = ((s + 1)\*(t + 1))/8;

dN7s = ((r + 1)\*(t + 1))/8;

dN7t = ((r + 1)\*(s + 1))/8;

dN8r = -((s + 1)\*(t + 1))/8;

dN8s = -((r - 1)\*(t + 1))/8;

dN8t = -((r - 1)\*(s + 1))/8;

dN=[[dN1r dN1s dN1t]' [dN2r dN2s dN2t]' [dN3r dN3s dN3t]'...

[dN4r dN4s dN4t]' [dN5r dN5s dN5t]' [dN6r dN6s dN6t]'...

[dN7r dN7s dN7t]' [dN8r dN8s dN8t]'];

end

Function defines derivatives of the node-less shape functions used to generate **B**:

function [dNa] = getdNa(r, s, t) %Computes dN for stiffness matrix calc.

% Shape function for incompatible modes:

dN9r = -2\*r;

dN9s = 0;

dN9t = 0;

dN10r = 0;

dN10s = -2\*s;

dN10t = 0;

dN11r = 0;

dN11s = 0;

dN11t = -2\*t;

dNa=[[dN9r dN9s dN9t]'...

[dN10r dN10s dN10t]'...

[dN11r dN11s dN11t]'];

end

Function defines isotropic stiffness matrix **E** used to generate **K**:

function [Em] = getE(E, v)

A=E/((1+v)\*(1-2\*v));

Em=A\*[1-v v v 0 0 0;

v 1-v v 0 0 0;

v v 1-v 0 0 0;

0 0 0 (1-2\*v)/2 0 0;

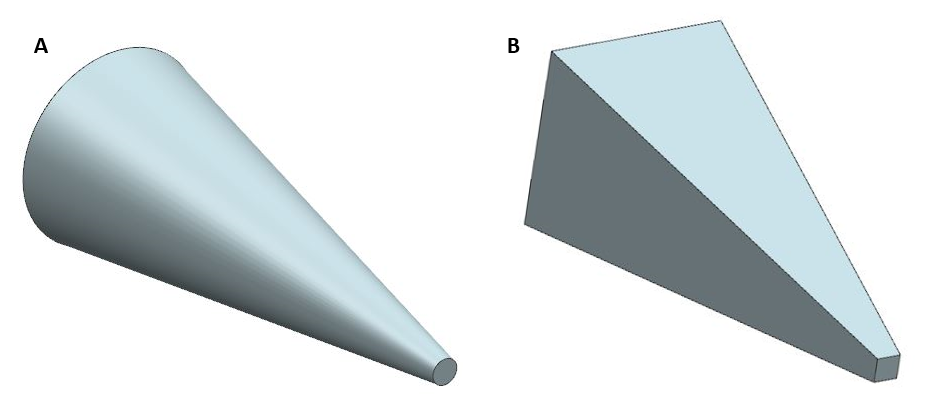
0 0 0 0 (1-2\*v)/2 0;

0 0 0 0 0 (1-2\*v)/2];

end

**Verification Studies**

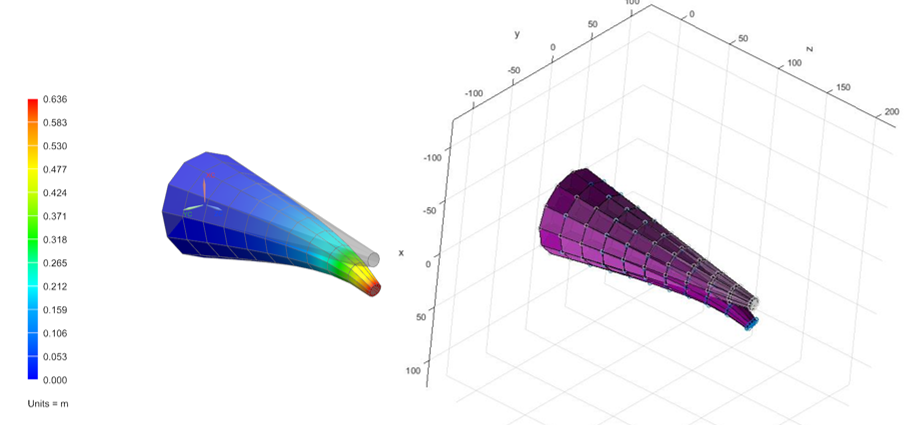
Models in Figure 4 were used to verify the element formulation. Two structures are a tapered cylinder and tapered pyramid.



**Figure 4**: A) Tapered cylinder. B) Tapered pyramid

The above two models were solved using Nastran brick element formulation and compared to the matlab brick formulation that was outlined above. Here are the following results for these studies.

**Tapered Cylinder results:**



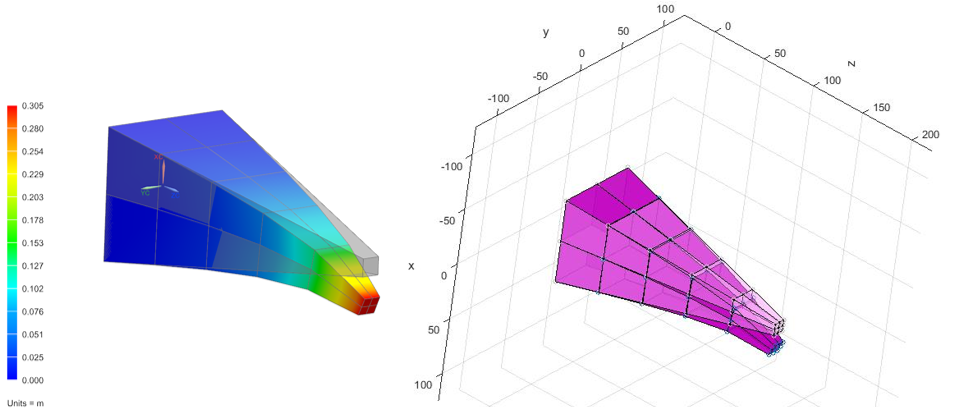
**Figure 5**: Deformation comparisons between Nastran and WFEM implementation.

Table 1 below lists the max tip displacement for the cylinder model.

|  |  |
| --- | --- |
|  | Max Tip Displacement (m) |
| **WFEM** | 0.667 |
| **Nastran** | 0.631 |

**Table 1**: Tapered cylinder max tip displacements

**Tapered Pyramid results:**



**Figure 6**: Deformation comparisons between Nastran and WFEM implementation.

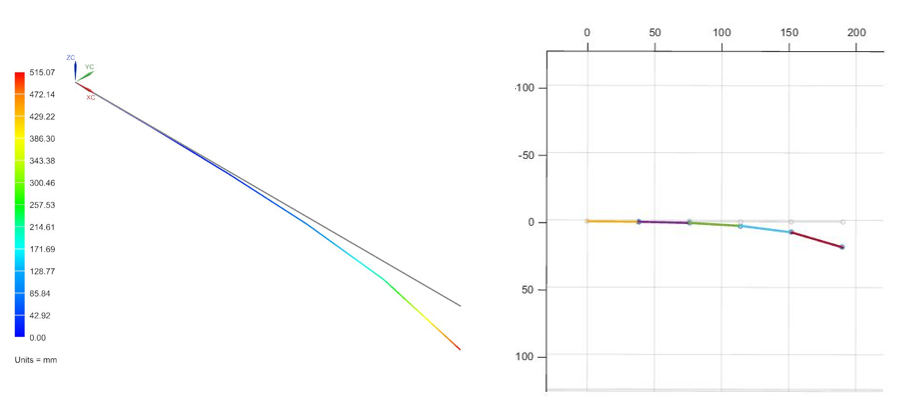
Table 2 below lists the max tip displacement for the pyramid model.

|  |  |
| --- | --- |
|  | Max Tip Displacement (m) |
| **WFEM** | 0.367 |
| **Nastran** | 0.305 |

**Table 2**: Tapered cylinder max tip displacements

In addition to the simulations performed using the brick elements. 1-D beam elements were used to further verify the brick element. Both Nastran and WFEM beam formulation from Project 1 was used. These results are listed below.

**Cylinder Beam Results:**



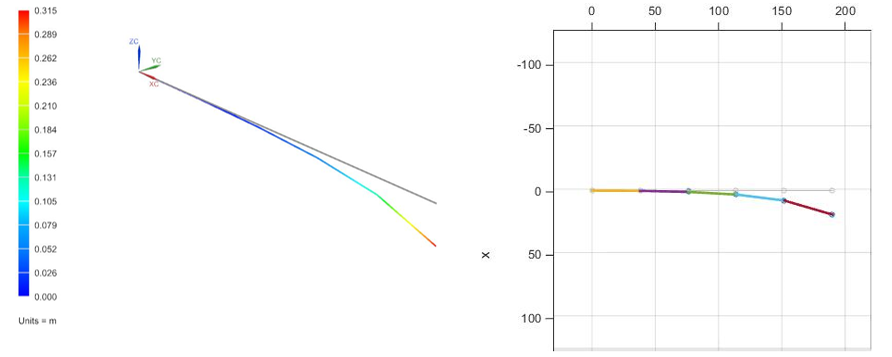
**Figure 7**: Tapered cylindrical beam results from Nastran and WFEM.

Table 3 below lists the max tip displacement for the cylinder brick and beam models

|  |  |
| --- | --- |
|  | Max Tip Displacement (m) |
| **WFEM Brick** | 0.667 |
| **Nastran Brick** | 0.631 |
| **WFEM Beam** | 0.467 |
| **Nastran Beam** | 0.515 |

**Table 3**: Brick and beam model tip displacements.

**Cylinder Beam Results:**



**Figure 8**: Tapered pyramid beam results from Nastran and WFEM.

Table 4 below lists the max tip displacement for the pyramid brick and beam models

|  |  |
| --- | --- |
|  | Max Tip Displacement (m) |
| **WFEM Brick** | 0.367 |
| **Nastran Brick** | 0.305 |
| **WFEM Beam** | 0.277 |
| **Nastran Beam** | 0.315 |

**Table 4**: Brick and beam model tip displacements.

In general tip displacements for the beam models are lower compared to the brick 3-D models. This can be attributed to the fact that the beam models are comprised of only 5 elements. If additional elements are added to the beam models the additional degrees of freedom will act to relax the response, leading to increased tip displacements. In general brick formulation implemented in WFEM agrees well with the Nastran results.