Beam Stress and Displacement Analysis

Alex Klein and Mark Daniel Leongomez
Johns Hopkins University
Mechanics Based Design
(for Professor Jaafar El-Awady)
Spring 2020

Project Goal

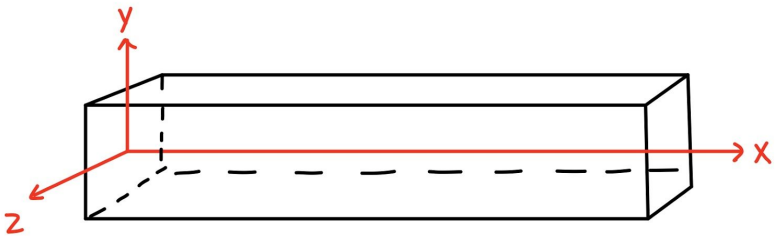
Given a beam of any material and shape, calculate the maximum stress and displacement due to any forces or point moments, constrained as a cantilever or by a pin and roller.

Completed Project

MATLAB program is available for download with this Google Drive link.
Download all files and open and run "*main*" to initiate the program.

Orientation

- Any number of forces of any magnitude can be applied to the beam in y and z directions for bending at any point along the x axis, as well as a single tensile or compressive force at the rightmost end of the beam.
- Any number of point moments can be applied along the length of the beam, about the y and/or z axis.
- Cantilever constraint scenario: the beam is fixed at the yz plane at x = 0.
- Pin and roller constraint scenario: the pin is along the z axis at x = 0 and there are two rollers, above and below the beam in the same direction as the pin, at a specific inputted spot on the beam along the x axis.



Coordinate System

Assumptions and Limitations

- While the height and width of the beam are accounted for in its moment of inertia, the beam is treated as a one dimensional line in order to use the singularity function method for displacement.
- The beam is assumed to be Euler-Bernouilli, is of uniform material, and has negligible weight in all calculations.
- The beam must be longer than it is wide or tall for the program to work, and the Euler-Bernouilli further restricts its slender ratio.
- The program is not intended for torsion on/about the x axis.
- Tensile and compressive forces can only be applied at the rightmost end of the beam only, where the resulting displacement is not included in the displacement outputs (only max stress calculations).
- Stress concentrations in the corners of the rectangular beam are ignored.
- The cantilever scenario prevents displacement at the yz plane, where the beam is assumed to be rigidly mounted.
- In the pin and roller constraint scenario, the pin prevents displacement deriving from forces in the x, y, and z directions, as well as moments about the x and y axis, but cannot prevent moments about the z axis. The roller only prevents up and down displacement in the y direction (at its specified x location) due to its reaction force.
- The Von Mises equation is used to calculate the max stress in the beam as a function of the x-position along the beam.

A note on one-dimensional assumption and Von Mises:
The Von Mises equation should be used to sum all the stresses acting upon a single point in the beam in order to find the equivalent yield stress if the point was in simple tension. In our simulation, the maximum shear stress due to bending at each section along the x axis (which should occur at the centroid), the maximum normal stress due to bending at this section (which should occur at the edge of the section), and the normal stress due to tension/compression are summed. This is then an overestimation of the max equivalent stress, as the max shear and normal do not occur at the same points but are being treated as if they do. This error should be small, however, as for Euler-Bernoulli beams of sufficient length the amount of shear stress due to bending is negligible.

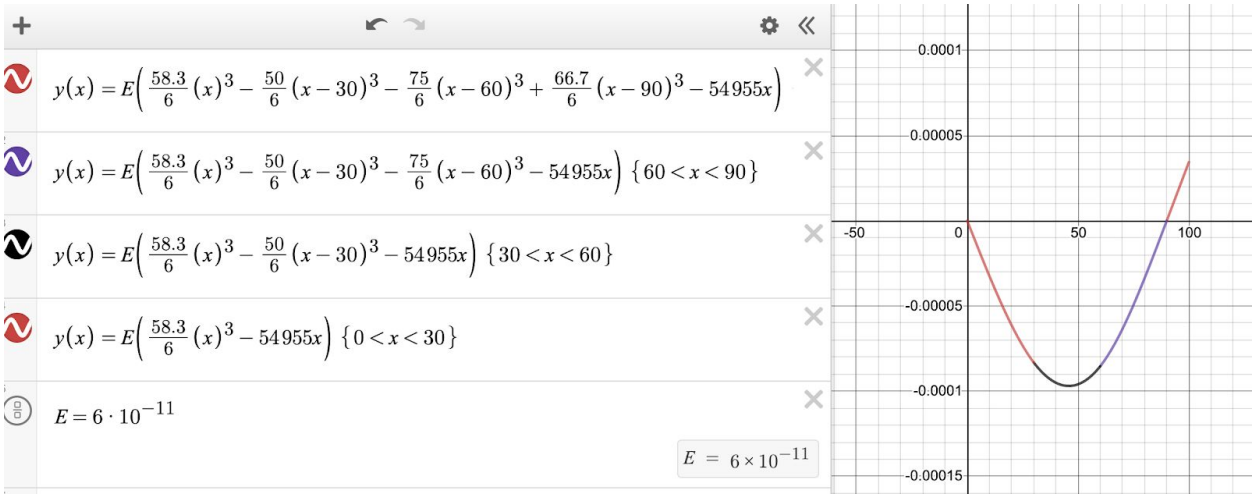
Implementation

Utilizing MATLAB, a main() script calls various other functions to complete the following steps:

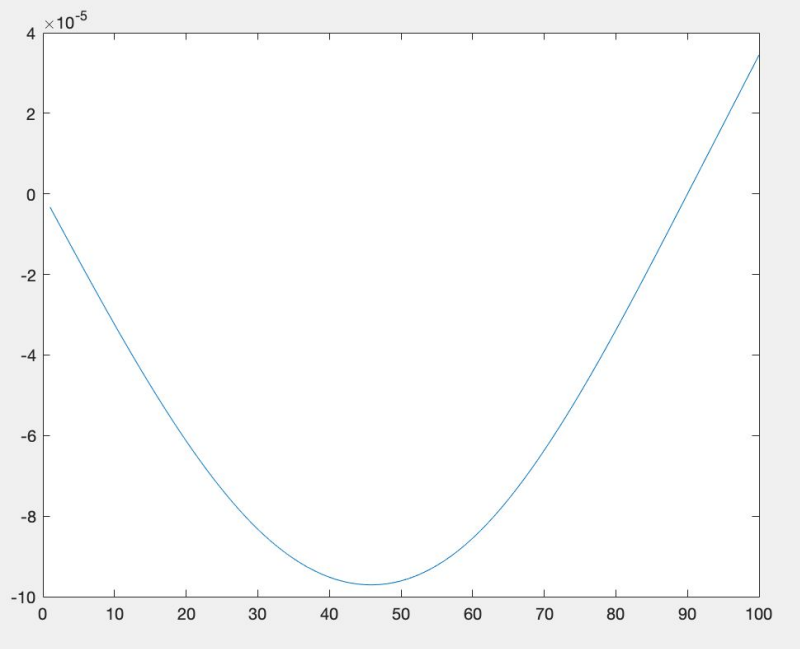
1. Input material, forces, moments, dimensions, constraint; Store as variables:
    a. Dimension array: [Lx, Ly, Lz]
    b. Force matrix: [Location(x), Fx, Fy, Fz] (multiple rows in matrix)
    c. Moment matrix: [Location(x), Mx, My, Mz]
    d. Constraints: 0 = fixed at x=0; # = distance of roller, hinge at x=0
2. Calculate simple intrinsic values (I / Q / E / G / v) from material input
3. Solve for reaction forces, put into matrix: [Rx1,Ry1,Rz1 ; Rx2,Ry2,Rz2]
4. Singularity integrate (from q to y) in both y and z directions
    a. Create load singularity function as matrix: q(y/z) = [magnitude, location, power]
    b. Singularity integrate through terms
    c. Repeat 3 more times for v, m, theta, y
5. Determine numerical boundary conditions and add to general integration solution
6. Calculate values from singularity functions and x position (n times in y and z directions)
    a. Shear stress
    b. Internal moment
    c. Displacement
7. Calculate normal stress from forces and poisson's ratio
8. Create (6 x 1001) matrix of all stresses
    a. Bending stress around z axis (in xy plane)
    b. Bending stress around y axis (in xz plane)
    c. Normal stress in x axis
    d. Shear stress due to bending about z axis (in xy plane)
    e. Shear stress due to bending around y axis (in xz plane)
    f. Shear stress due to torsion around x axis
9. Calculate total resultant stress with Von Mises equation
10. Find max stress and displacement over entire beam from matrices
11. Output values and visualize

Validation

To ensure the program is working properly, we calculated multiple test cases by hand and found several textbook example problems. With the solutions available, we graphed the piecewise singularity functions in Desmos and ran the corresponding inputs through the program. We confirmed that the MATLAB graphic output was consistent with the Desmos plots in both y and z displacement directions. Similarly for maximum stresses, we worked problems by hand as well as comparing example textbook problems to the Von Mises stress outputs.



Desmos Plot



Corresponding MATLAB Program Output (displacement in one direction)

Example Run and Output

The example run tests a 3m long, 4cm tall, and 3cm wide 4041 alloy steel beam that is constrained by a pin at x=0mm and roller at x=2m. A force of 50N is applied in the negative y direction at the end of the beam x=3m, as well as a positive z force of 25N at x=1m, and a tensile force of 75N at the end of the beam. There are no point moments.

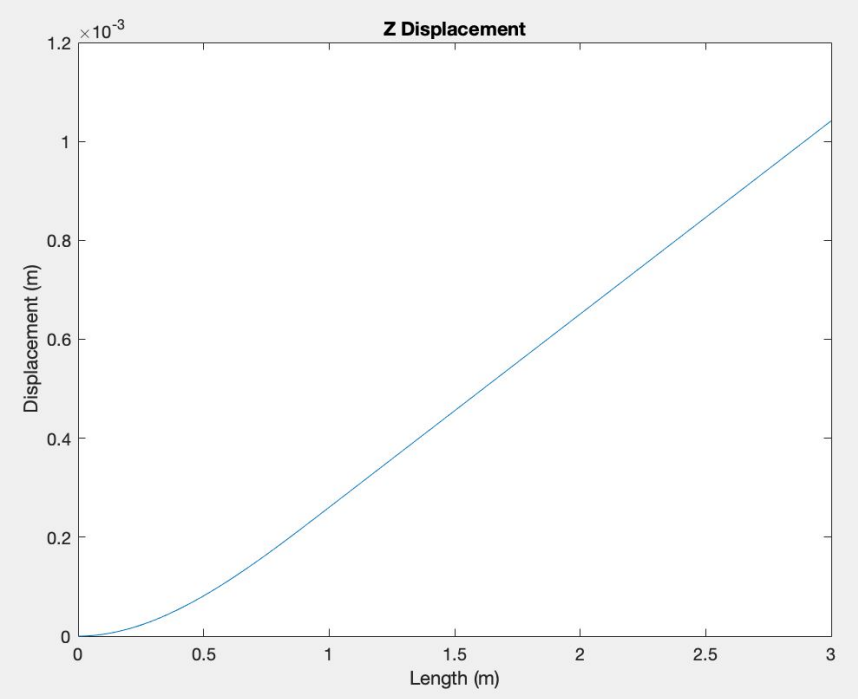
Input Code:

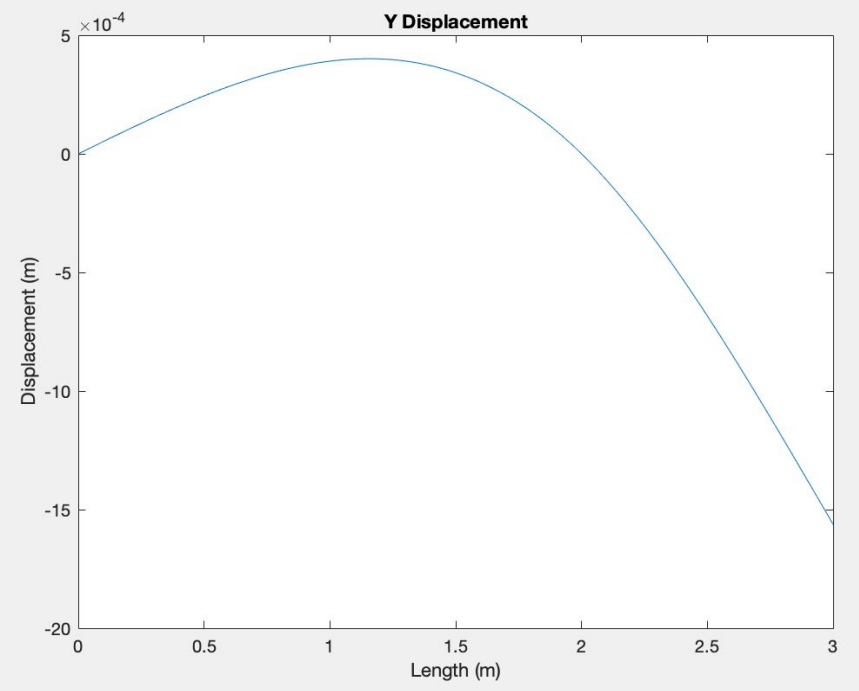
```
>> main

Specify beam material (enter number):
1 for Rigid PVC
2 for 6061-T6 Aluminum
3 for 4041 Alloy Steel
4 for Wood (Red Maple)
5 for Copper Alloy
6 Enter custom properties
3

Enter beam length [x direction] (m): 3
Enter beam height [y direction] (m): 0.04
Enter beam width [z direction] (m): 0.03

Specify type of beam constraint.
Enter 0 for fixed/cantilever.
Enter 1 for hinge/roller.
1
Enter roller distance from end (m): 2

How many forces do you want to apply to the beam: 2
Force 1 x location (m): 3
Input force 1 y and z components (N):
Fy= -50
Fz= 0
Force 2 x location (m): 1
Input force 2 y and z components (N):
Fy= 0
Fz= 25

Normal force on end of beam [along x axis] (N): 75

How many point moments do you want to apply to the beam: 0
```

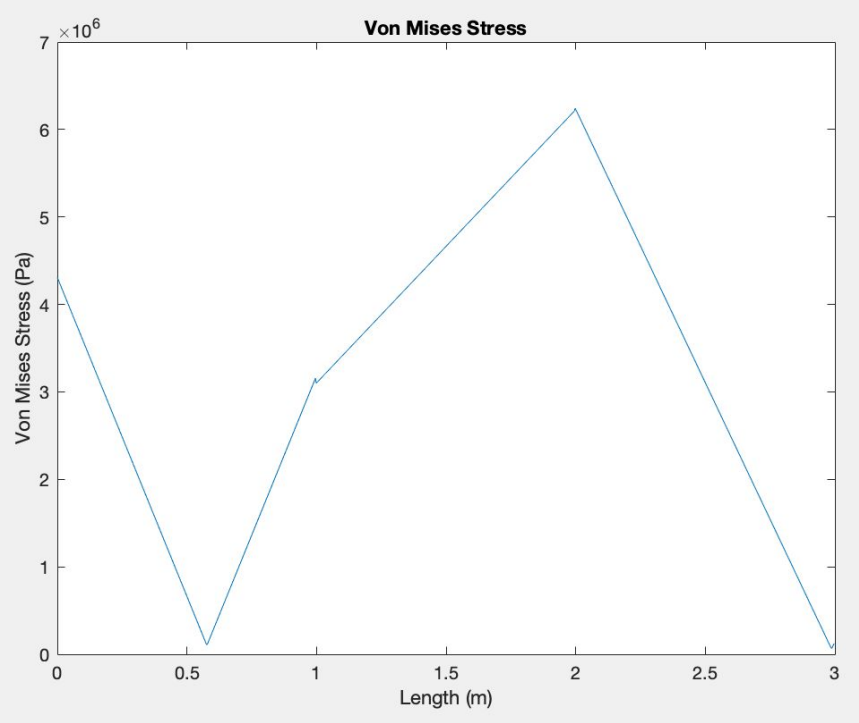
Output Text:

The maximum displacement in the Y direction is 1.56e-03 m at x = 3.00 m
The maximum displacement in the Z direction is 1.04e-03 m at x = 3.00 m
The maximum Von Mises Stress in the beam is 6.24e+06 Pa at x = 2.00 m

Output Plots:







Appendix: Program Code

1. main.m
2. createLoadSingularity.m
3. findMAX.m
4. Graphs.m
5. inputs.m
6. integrateQ.m
7. momentOfInertia.m
8. normalStress.m
9. qBeam.m
10. reactionForces.m
11. sFuncIntegrate.m
12. stepTerm.m
13. stressCalc.m
14. valueAtSection.m
15. VonMisesStress.m

```
function main()

% inputs
[material, length, constraint, force, moment] = inputs();

% calculate moment of inertia and Q
I = momentOfInertia(length);
Q = qBeam(length);

% create load singularity function (q)
reactions = reactionForces(constraint, force, moment);
[qy,qz] = createLoadSingularity(force, moment, reactions, constraint);

% integrate to find v,m,y functions (with boundary conditions)
[my,vy,yy] =
 integrateQ(qy,constraint,force,reactions,2,material(1,1),I);
[mz,vz,yz] =
 integrateQ(qz,constraint,force,reactions,3,material(1,1),I);

% determine moment, shear, displacement in beam
myMatrix = valueAtSection(my,length);
vyMatrix = valueAtSection(vy,length);
yyMatrix = valueAtSection(yy,length);
mzMatrix = valueAtSection(mz,length);
vzMatrix = valueAtSection(vz,length);
yzMatrix = valueAtSection(yz,length);

% stresses and von mises
normal = normalStress(force,length);
stress =
 stressCalc(normal,mzMatrix,myMatrix,vzMatrix,vyMatrix,length,I,Q);
VMstress = VonMisesStress(stress);

% output
Graphs(yyMatrix,yzMatrix,VMstress,length);
findMAX(yyMatrix,yzMatrix,VMstress,length);

end
```

Published with MATLAB® R2018b

```
function [qy,qz] = createLoadSingularity(force, moment, reactions,
 constraint)

% create q(y) matrix
qy = zeros(1,3);
for i=1:size(force,1)
   qy = [qy; [force(i,3),force(i,1),-1]];
end
for i=1:size(moment,1)
   qy = [qy; [moment(i,4),moment(i,1),-2]];
end
qy = [qy; [reactions(2,1),0,-1]];
if constraint == 0
    qy = [qy; [-reactions(6,1),0,-2]];
else
    qy = [qy; [reactions(6,1),constraint,-1]];
    qy = qy(2:end,:);
end

% create qz matrix
qz = zeros(1,3);
for i=1:size(force,1)
   qz = [qz; [force(i,4),force(i,1),-1]];
end
for i=1:size(moment,1)
   qz = [qz; [moment(i,3),moment(i,1),-2]];
end
qz = [qz; [reactions(3,1),0,-1]];
qz = [qz; [-reactions(5,1),0,-2]];
qz = qz(2:end,:);

end
```

Published with MATLAB® R2018b

```
function findMAX(yyMatrix,yzMatrix,VMstress,length)

[MaxY,MaxYloc] = max(abs(yyMatrix));
[MaxZ,MaxZloc] = max(abs(yzMatrix));
[MaxVM,MaxVMloc] = max(abs(VMstress));

fprintf('--- \n');

if MaxY == 0
    fprintf('There is no displacement in the Y direction\n')
else
    fprintf('The maximum displacement in the Y direction is %.2d m at
 x = %.2f m \n', MaxY, (MaxYloc-1)/1000*length(1,1))
end

if MaxZ == 0
    fprintf('There is no displacement in the Z direction\n')
else
    fprintf('The maximum displacement in the Z direction is %.2d m at
 x = %.2f m \n', MaxZ, (MaxZloc-1)/1000*length(1,1))
end

fprintf('The maximum Von Mises Stress in the beam is %.2d Pa at x =
 %.2f m \n', MaxVM, (MaxVMloc-1)/1000*length(1,1))

end
```

Published with MATLAB® R2018b

```
function Graphs(yyMatrix,yzMatrix,VMstress,length)

figure(1)
plot([0:length(1,1)/1000:length(1,1)],yyMatrix);
title('Y Displacement');
xlabel('Length (m)');
ylabel('Displacement (m)');

figure(2)
plot([0:length(1,1)/1000:length(1,1)],yzMatrix);
title('Z Displacement');
xlabel('Length (m)');
ylabel('Displacement (m)');

figure(3)
plot([0:(length(1,1)/1000):length(1,1)-1/1000],VMstress(1,2:end));
title('Von Mises Stress');
xlabel('Length (m)');
ylabel('Von Mises Stress (Pa)');

end
```

Published with MATLAB® R2018b

```
function [material, length, constraint, force, moment] = inputs()

fprintf('\n');

% MATERIAL
fprintf('Specify beam material (enter number): \n');
fprintf ('1 for Rigid PVC \n');
fprintf ('2 for 6061-T6 Aluminum \n');
fprintf ('3 for 4041 Alloy Steel \n');
fprintf ('4 for Wood (Red Maple) \n');
fprintf ('5 for Copper Alloy \n');
fprintf ('6 Enter custom properties \n');
n = input('');
switch n
    case 1
        material = [3000000000,1000000000,0.4];
    case 2
        material = [70000000000,26000000000,0.33];
    case 3
        material = [200000000000,80000000000,0.28];
    case 4
        material = [10000000000,3000000000,0.43];
    case 5
        material = [117000000000,45000000000,0.34];
    case 6
        material(1,1) = input('Enter Modulus of Elasticity (Pa) E =
 ');
        material(1,2) = input('Enter Modulus of Rigidity (Pa) G = ');
        material(1,3) = input('Enter Poisson''s Ratio v = ');
end
fprintf('\n');

% LENGTH
length = zeros(1,3);
length(1,1) = input('Enter beam length [x direction] (m): ');
length(1,2) = input('Enter beam height [y direction] (m): ');
length(1,3) = input('Enter beam width [z direction] (m): ');
fprintf('\n');

% CONSTRAINT
fprintf('Specify type of beam constraint. \nEnter 0 for fixed/
cantilever. \nEnter 1 for hinge/roller. \n');
constraint = input('');
if constraint == 1
    constraint = input('Enter roller distance from end (m): ');
end
fprintf('\n');

% FORCES
n = input('How many forces do you want to apply to the beam: ');
force = zeros(n,4);
for i=1:n
```

```
    %location
    fprintf('Force %d x location (m): ',i);
    force(i,1) = input('');
    %components
    fprintf('Input force %d y and z components (N):',i);
    fprintf('\n');
    force(i,2) = 0;
    force(i,3) = input('Fy= ');
    force(i,4) = input('Fz= ');
end
fprintf('\n');

% NORMAL FORCE
x = input('Normal force on end of beam [along x axis] (N): ');
force = [[0,x,0,0];force];
fprintf('\n');

% POINT MOMENT
n = input('How many point moments do you want to apply to the beam:
 ');
moment = zeros(n+1,4);
for i=2:n+1
    %location
    fprintf('Moment %d x location (m): ',i-1);
    moment(i,1) = input('');
    %components
    fprintf('Input moment %d about the y and z axis (Nm):',i-1);
    fprintf('\n');
    moment(i,2) = 0;
    moment(i,3) = input('My= ');
    moment(i,4) = input('Mz= ');
end
fprintf('\n');

% TORSION
moment(1,2) = 0;
%moment(1,2) = input('Torsion on beam (about x axis in Nm): ');
%moment(1,1) = length(1,1);
%fprintf('\n');

end
```

Published with MATLAB® R2018b

```
function [m,v,y] = integrateQ(q,constraint,force,reactions,dim,E,I)

% integrate
v = sFuncIntegrate(q);
m = sFuncIntegrate(v);
thetaGEN = sFuncIntegrate(m);

% boundary condition
% C1 and C2 = 0 for fixed
% C2 = 0 for hinge/roller

C = 0;

% find C1 for hinge/roller:
if constraint ~= 0 && dim == 2
    C=reactions(2,1)*constraint^3;
    for i=1:size(force,1)
        if constraint > force(i,1)
        C=C+force(i,dim+1)*(constraint-force(i,1))^3;
        end
       % where dim is 1=x,2=y,3=z based on function dimension
    end
    C=-C/(6*constraint);
end

% add boundary condition
theta = [thetaGEN(:,1)/(E*I(1,3)),thetaGEN(:,2:3);[C/(E*I(1,3)),0,0]];
y = sFuncIntegrate(theta);

end
```

Published with MATLAB® R2018b

```
function inertia = momentOfInertia(length)

inertia = zeros(1,3);
inertia(1,1) =
 (1/12)*length(1,2)*length(1,3)*(length(1,2)^2+length(1,3)^2);
inertia(1,2) = ((length(1,2)*length(1,3)^3)/12);
inertia(1,3) = (length(1,2)^3*length(1,3)/12);

end
```

Published with MATLAB® R2018b

```
function normal = normalStress(force,length)

xForce = force(1,2);
area = length(1,2)*length(1,3);

normal = xForce/area;

end
```

Published with MATLAB® R2018b

```
function Q = qBeam(length)

%calcalates Q for both bending directions of the beam
Q(1,1) = 0;
Q(1,2) = length(1,2)^2*length(1,3)/8;
Q(1,3) = (length(1,2)*length(1,3)^2)/8;

end
```

Published with MATLAB® R2018b

```
function [reactions] = reactionForces(constraint, force, moment)

% define coefficient matrix
if constraint == 0 %cantilever
    Rm = -[0,0,0;0,0,0;0,0,0;1,0,0;0,1,0;0,0,1];
    R = -[1,0,0;0,1,0;0,0,1;0,0,0;0,0,0;0,0,0];
elseif constraint > 0 %hinge/roller
    Rm = -[0,0,0;0,0,1;0,0,0;1,0,0;0,1,0;0,0,constraint];
    R = -[1,0,0;0,1,0;0,0,1;0,0,0;0,0,0;0,0,0];
else %error
    fprintf('error with reactionForces function');
end
Coeff = [R,Rm];

% define equilibrium vector
Equil = zeros(6,1);
Equil(4,1) = moment(1,1);
Equil(5,1) = moment(1,2);
Equil(6,1) = moment(1,3);
for i=1:size(force,1)
    Equil(1,1) = Equil(1,1) + force(i,2);
    Equil(2,1) = Equil(2,1) + force(i,3);
    Equil(3,1) = Equil(3,1) + force(i,4);
    Equil(4,1) = Equil(4,1);
    Equil(5,1) = Equil(5,1) + force(i,4)*force(i,1);
    Equil(6,1) = Equil(6,1) + force(i,3)*force(i,1);
end

% solve for reactions vector (6,1)
reactions = Coeff\Equil;

end
```

Published with MATLAB® R2018b

```
function sFunc2 = sFuncIntegrate(sFunc1)

% takes each term of sFunc matrix, integrates it, reconcatenates

sFunc2 = zeros(1,3);
terms = size(sFunc1,1);

for i=1:terms
    sFunc2(i,:) = stepTerm(sFunc1(i,:));
end

end
```

Published with MATLAB® R2018b

```
function newTerm = stepTerm(sFuncTerm)

% stepTerm and sFuncTerm are row vectors with
    % magnitude
    % location
    % power
% of singularity function term

magnitude = sFuncTerm (1,1);
location = sFuncTerm (1,2);
power = sFuncTerm (1,3);

if power <= 0
    newTerm = [magnitude, location, power+1];
end

if power >= 1
    newTerm = [magnitude/(power+1), location, power+1];
end

end
```

Published with MATLAB® R2018b

```
function stress = stressCalc(stressNorm, mzMatrix, myMatrix, vzMatrix,
 vyMatrix, length, I, Q)

stress = zeros(6,100);

for j = 1:1001

 stress(2,j) = (mzMatrix(1,j)) * length(1,3)/(I(2) * 2);
 stress(1,j) = (myMatrix(1,j)) * length(1,2)/(I(3) * 2);
 stress(3,j) = stressNorm;
 stress(5,j) = vzMatrix(1,j)*Q(1,3)/(I(1,2) * length(1,2));
 stress(4,j) = vyMatrix(1,j)*Q(1,2)/(I(1,3) * length(1,3));
 stress(6,j) = 0;

end

end
```

Published with MATLAB® R2018b

```
function valueMatrix = valueAtSection(inputFunction,length)

valueMatrix = zeros(1,1);
i=1;

for position = 0:length(1,1)/1000:length(1,1)

    value = 0;
    for term = 1:size(inputFunction)
        if inputFunction(term,3) >=0
            if position > inputFunction(term,2)
                value = value + inputFunction(term,1)*(position-
inputFunction(term,2))^(inputFunction(term,3));
            end
        end
    end

    valueMatrix(1,i) = value;
    i = i+1;

end

end
```

Published with MATLAB® R2018b

```
function VMstress = VonMisesStress(stress)

VMstress = zeros(1,1001);

for i = 1:1001

    x = stress(1,i) + stress(2,i) + stress(3,i);
    xy = stress(4,i);
    xz = stress(5,i);
    yz = stress(6,i);

    VMstress(i) = abs(x) + (3*((xy^2)+(xz^2)+(yz^2)))^0.5;

end

end
```

Published with MATLAB® R2018b