Mini Project

Stresses in Beams

Team:

Meet Gandhi 15110049

Darshan Patel 15110083

Rajat Ranjan 15110098

Saksham Singal 15110112

Table of Contents

Introduction: ... 2
Why this project? ... 2
Theory: ... 2
  Introduction ... 2
  Some basic concepts ... 2
  Pure Bending Scenario ... 4
  Deriving the flexural formula ... 4
  Using the flexural formula ... 6
Working of the Software ... 6
  About the Code ... 6
  About the GUI ... 9
User Manual for the Software ... 9
Validation and test cases: ... 14
  Symmetrical Bending (Moment acting in 1 direction) ... 14
  Symmetrical Bending (Moment acting in 2 directions) ... 22
Limitations of the software ... 29
References: ... 30
List of Figures: ... 31
Appendix ... 32
  Code for Test Function ... 32
  Code for Polygeon Function ... 34
  Code used for GUI ... 36
  Code for input gui Function ... 36
  Code for page_2_gui Function (Page 2 of GUI) ... 48
  Code for page_3_gui Function (Page 3 in GUI) ... 53
  Code for Stress_point Function ... 60
  Code for Stress_calculator Function ... 62

Introduction:

The project deals with stresses in beams or more precisely, the stresses developed on the cross section of beam. The following report contains in depth theoretical knowledge of the subject starting from the beginner level and moving on to more advanced scenarios. This document also serves as a user manual to the software that has been developed to compute stresses in beams for various cases. The algorithms and methodology used for developing the software has been highlighted in the sections that follow.

Why this project?

The motive of this project is to impart a holistic understanding of the subject to the readers whilst at the same time providing them with a tool to run test cases to get an intuitive feel of the subject. The software can also be used as verification tool in various situations. Improving the algorithms and inculcating varied scenarios over time would make the software more vibrant in terms of the situations it can tackle and more precise in its output.

Theory:

Introduction

A beam whether hinged, simply supported or unsupported develops a stress when subjected to a force or moment. The magnitude and direction of the stress depends on external conditions such as hinging, supports, location of force applied etc. as well as on the geometry of the cross section, material of the beam etc. The topic of stresses in beams can be broken down into 2 parts namely pure bending scenarios and scenarios with shear forces involved. This project is based on the pure bending scenario which will constitute the major part of the literature.

Some basic concepts

We start off by defining some concepts that are prerequisites to understanding the stress distribution and formulas involved.

1. **Neutral Axis:** As a beam undergoes pure bending, some longitudinal sections witness compression and some witness tension as can be seen from *Figure 1*. The axis that undergoes compression reduced in length whereas the axes that undergo tension

increase in length. Neutral axis is defined as that axis whose length remains same post deformation.

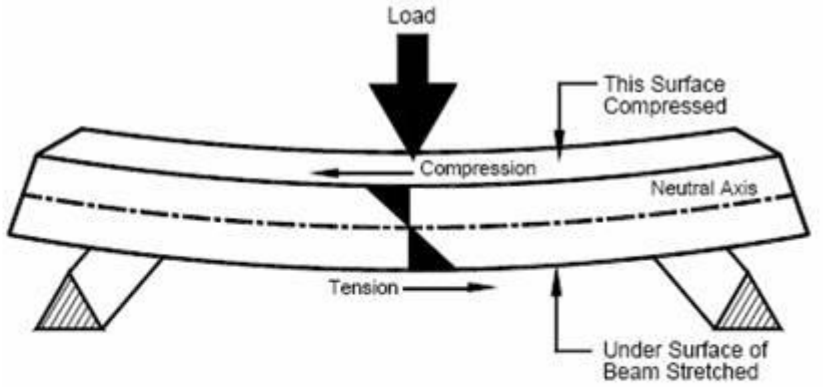


Figure 1: Compression and Tension in a Beam[2]

2. **Moment of Inertia:** Moment of inertia is the analogous to mass in linear motion[3]. It is defined for various shapes and structures. The moment of inertia for a point mass is given by

$$I=M*R^2$$

Using the above formula and with the help of integration, Moment of Inertias of various shapes can be found out. Moment of Inertias of some common shapes are mentioned below.

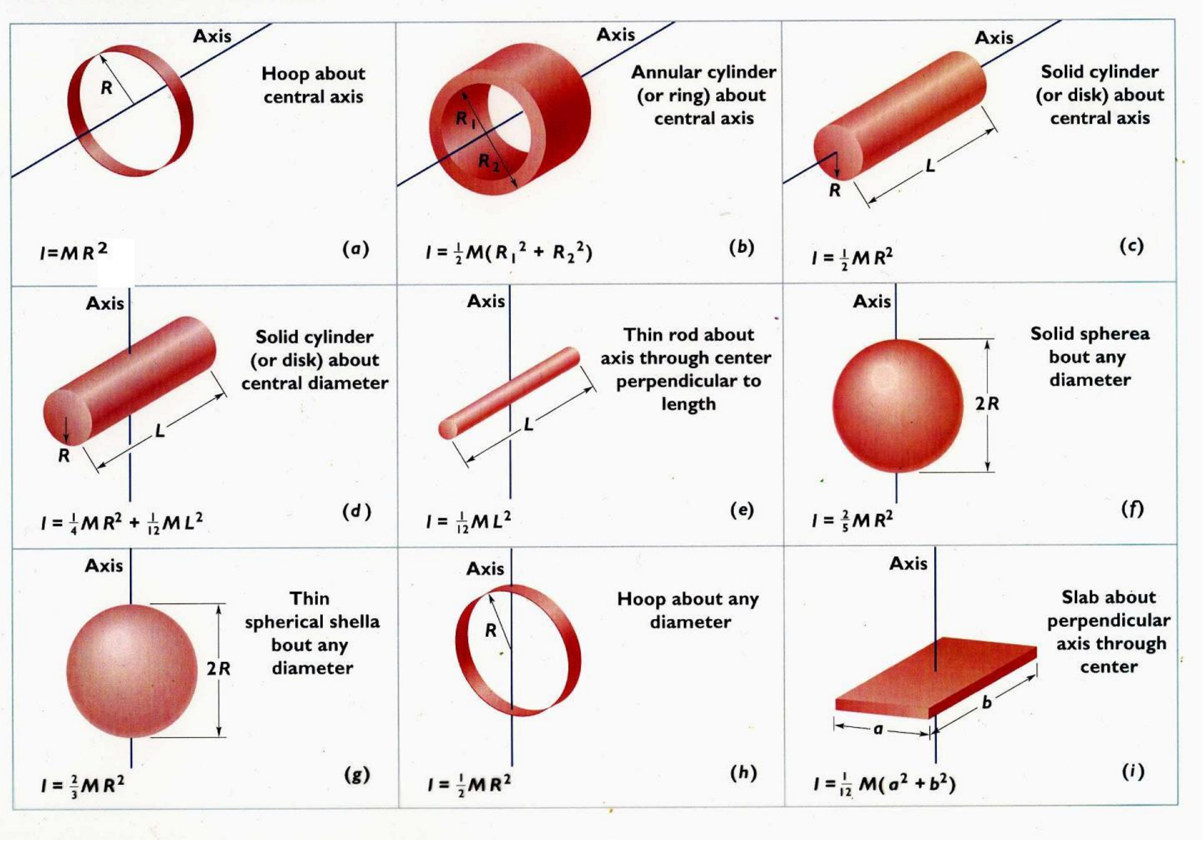


Figure 2: Moment of Inertia of some common shapes[4]

Pure Bending Scenario

A beam is said to be in pure bending when it is subjected to a constant bending moment only. In other words, there is no shear force acting on the cross-sectional surface. Before deriving the formula for stress in the beams, we have to make the following assumptions[5].

- Plane cross sections remain plane after loading.
- The cross sections are uniform throughout the beam
- The modulus of elasticity remains the same for both tension and compression.
- Shear centre passes through the centroidal axis.

Deriving the flexural formula[6][7]

The following diagram represents a schematic of a beam that is subjected to pure bending.

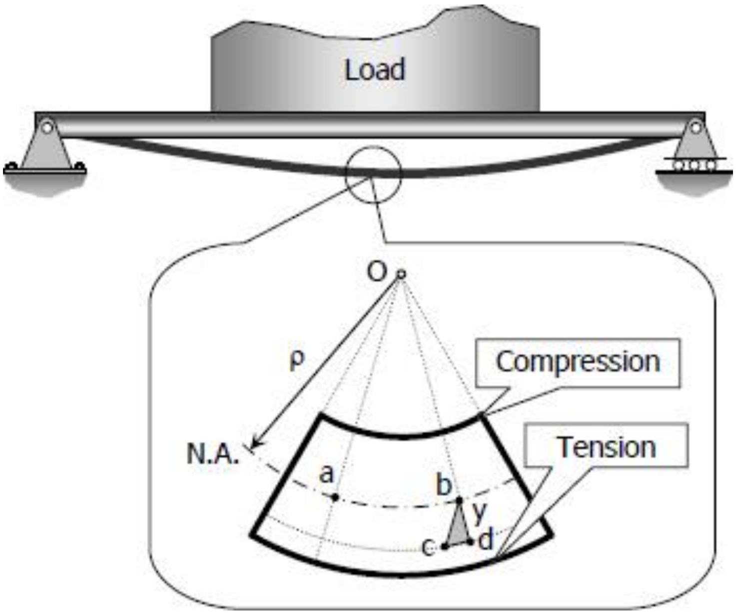


Figure 3 Bending in a beam

Consider Fig.4 which show the longitudinal section of a beam. We start off by writing the expression for strain in the beam.

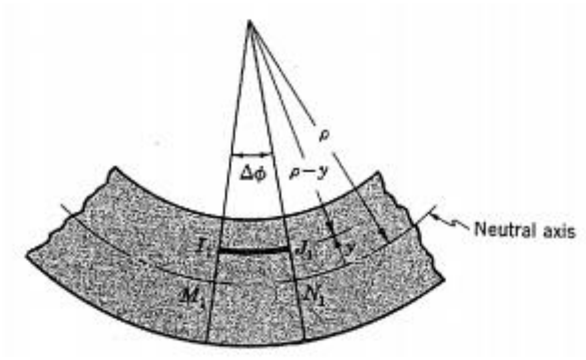


Figure 4 Longitudinal cross section of a beam

From the figure,

$$Ex=(IJ-MN)/MN$$

Where MN is the undeformed length on the neutral axis and IJ is the deformed length.

Now, $MN= \rho*\Delta\phi$

And $IJ= (\rho-y)*\Delta\phi$

Substituting in the equation for Ex we get

$$E_x = -y/ \rho$$

Now that we have got the expression for strain, we can calculate the stress. Consider the following figure.

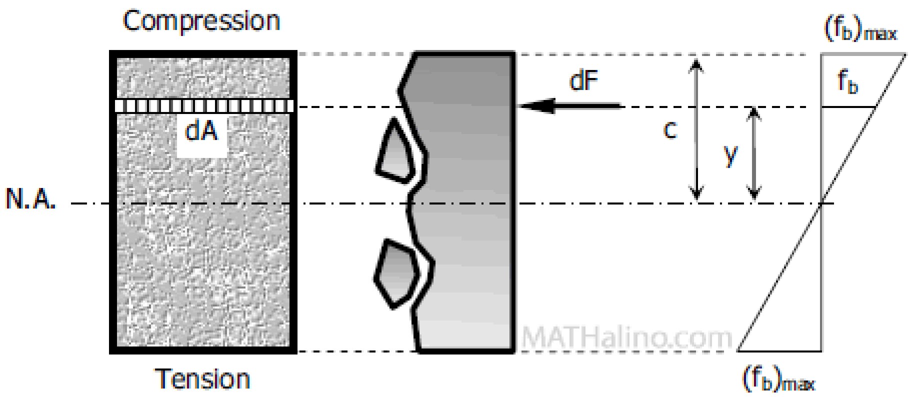


Figure 5 Derivation of stresses in beams

Let dF be the force acting on a small element of area dA at a distance y from the neutral axis.

Stress = dF/dA =Modulus of Elasticity* Strain

Therefore, $$dF/dA=E*(-y/ \rho)$$

$$dF=dA*E*(-y/ \rho)$$

For the beam to be in equilibrium, the moment about the neutral axis must be balanced.

Hence,

$$dM=y*dF$$

$$dM=y*dA*E*(-y/ \rho)$$

Integrating dM we get

$$M=-E/ \rho\int(y^2*dA)$$

$$\int(y^2*dA)=I_x$$

$$M/I_x=-E/ \rho$$

Multiplying both sides with y we get

$$M*y/I_x=E(-y/\rho)$$

$$\sigma_z=My/I_x$$

The following formula is applicable when the loading is symmetric and the axes used are centroidal principal axes.

For bending in which moment is acting along the 2 principal directions we get the formula by superimposing the two values in the two directions

$$\sigma_z = M_y*x/I_y$$

$$\sigma_z = -M_x*y/I_x$$

$$\sigma_z = -M_x*y/I_x + M_y*x/I_y$$

In both the cases described above, the axes were taken along the principal directions which results in the I_{xy} component of Moment of Inertia to be 0.

Considering a case when the axes are not along the principal directions, we get

$$\sigma_z=(M_yI_x + M_xI_{xy})*x/(I_x I_y - I^2_{yx}) + (M_xI_y + M_yI_{yx})*y/(I_x I_y - I^2_{yx})$$

Using the flexural formula

The following steps can be used to easily solve for stresses in beams.

1. Find out the moment at the desired cross section using Bending Moment Diagram.
2. Find the centroid of the given section and shift origin to that point
3. Check whether the given cross section is symmetric or unsymmetric.about our chosen axes
4. If the cross section is symmetric, use the first formula to calculate stress distribution.
5. If the moment is acting in 2 directions, align one of the axis to the line of symmetry. This gives us the principal directions
6. Calculate the moment of inertias and use the superimposition bending formula.
7. You can also directly use the general flexural formula if you do not know the principal directions.

Working of the Software

In this section we will go through the methodology used to execute the working of the software. We start off by describing the working of the code followed by a discussion on the working of the GUI and visualization.

About the Code

The code can be broken down into 3 part each of which has been discussed in detail below.

- **Input**

Before calculating the stresses on the cross section, the code must understand the type of shape we are working with. To accomplish the same, the user is asked to input the end points of the cross section in counter-clockwise order. This helps the code build the 2D structure whose moment of inertia is to be calculated. This method works for straight edge scenarios and not curved edges. For common geometries, only a few parameters have to be specified to determine the shape. So, instead of inputting all the end points, only the essential parameters are taken as inputs. The end points of the cross section are then calculated based on the parameters input by the user. The details of how to enter inputs have been mentioned in the user manual.

After defining the geometry of the cross section, the code also requires additional parameters such as moments in x and y directions and the coordinates of the point at which the value of stress is required.

- **Processing**

At the program end, the input coordinates are passed on as two vectors, one for x coordinates and the other of all y coordinates. These are passed to the function ‘ POLYGEOM’[8]. This function calculates the area of the cross section, coordinates of the centroid, perimeter and area moment of inertias about the given axes and the centroidal principal axes. The angle that the principal axes make with the given axes is also calculated.

The flow of operations in the program has been described below:

1. The first step is to check if the inputs entered are valid. Hence the length of both x and y vectors are verified to be equal.
2. Now the area, coordinates of centroid, moment of inertias, perimeter and angles are calculated based on the formulas described below[9]:
    a. $a_i = x_i y_{i+1} - x_{i+1} y_i$
    b. $A = \frac{1}{2}\sum_1^n a_i$
    c. $x_c = \frac{1}{6A}\sum_1^n a_i (x_i + x_{i+1})$
    d. $y_c = \frac{1}{6A}\sum_1^n a_i (y_i + y_{i+1})$
    e. $I_{xx} = \frac{1}{12}\sum_1^n a_i \left(y_i^2 + y_i y_{i+1} + y_{i+1}^2\right)$
    f. $I_{yy} = \frac{1}{12}\sum_1^n a_i \left(x_i^2 + x_i x_{i+1} + x_{i+1}^2\right)$
    g. $I_{xy} = \frac{1}{24}\sum_1^n a_i (x_i y_{i+1} + 2x_i y_i + 2x_{i+1} y_{i+1} + x_{i+1} y_i)$

h. $\text{perimeter} = \sum_{1}^{n} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$

Centroidal moments

i. $I_{uu} = I_{xx} - Ay_c^{\,2}$ $I_{vv} = I_{yy} - Ax_c^{\,2}$ $I_{uv} = I_{xy} - Ax_c y_c$ $J = I_{uu} + I_{vv} = I_1 + I_2$

Principal moments

j. $I_1, I_2 = (I_{uu} + I_{vv})/2 \pm \sqrt{(I_{uu} + I_{vv})^2/4 + I_{uv}^{\,2}}$ $\tan 2\theta = 2I_{uv}/(I_{vv} - I_{uu})$

k. $I_1, I_2 = \text{eig}\begin{bmatrix} I_{uu} & -I_{uv} \\ -I_{uv} & I_{vv} \end{bmatrix}$

The formulas mentioned above are obtained using Green's Theoram where the area integral is converted into boundary integral and then discretized using the following table:

QUANTITY	AREA INTEGRAL $\iint \left(\frac{\partial B}{\partial y} - \frac{\partial C}{\partial x}\right) dx\,dy$	CLOCKWISE BOUNDARY INTEGRAL $\oint (B\,dx + C\,dy)$	CLOCKWISE SUMMATIONS FOR CLOSED POLYGON n vertices $x_i\ y_i$ $\Delta x = x_{i+1} - x_i$ $\Delta y = y_{i+1} - y_i$ $x_{n+1} = x_1$ $y_{n+1} = y_1$
Area A	$\iint dx\,dy$	$\oint (y\,dx - x\,dy)/2$	$\sum_{i=1}^{n} (y_i \Delta x - x_i \Delta y)/2$
First Moment $A x_c$ about y axis	$\iint x\,dx\,dy$	$\oint (2xy\,dx - x^2\,dy)/4$	$\sum_{i=1}^{n} (6x_i y_i \Delta x - 3x_i^2 \Delta y + 3y_i \Delta x^2 + \Delta x^2 \Delta y)/12$
First Moment $A y_c$ about x axis	$\iint y\,dx\,dy$	$\oint (y^2\,dx - 2xy\,dy)/4$	$\sum_{i=1}^{n} (3y_i^2 \Delta x - 6x_i y_i \Delta y - 3x_i \Delta y^2 - \Delta x \Delta y^2)/12$
Second Moment I_{xx} about x axis	$\iint y^2\,dx\,dy$	$\oint (y^3\,dx - 3xy^2\,dy)/6$	$\sum_{i=1}^{n} (2y_i^3 \Delta x - 6x_i y_i^2 \Delta y - 6x_i y_i \Delta y^2 - 2x_i \Delta y^3 - 2y_i \Delta x \Delta y^2 - \Delta x \Delta y^3)/12$
Second Moment I_{yy} about y axis	$\iint x^2\,dx\,dy$	$\oint (3x^2 y\,dx - x^3\,dy)/6$	$\sum_{i=1}^{n} (6x_i^2 y_i \Delta x - 2x_i^3 \Delta y + 6x_i y_i \Delta x^2 + 2y_i \Delta x^3 + 2x_i \Delta x^2 \Delta y + \Delta x^3 \Delta y)/12$
Cross Moment I_{xy}	$\iint xy\,dx\,dy$	$\oint (xy^2\,dx - x^2 y\,dy)/4$	$\sum_{i=1}^{n} (6x_i y_i^2 \Delta x - 6x_i^2 y_i \Delta y + 3y_i^2 \Delta x^2 - 3x_i^2 \Delta y^2 + 2y_i \Delta x^2 \Delta y - 2x_i \Delta x \Delta y^2)/24$
Perimeter P		$\oint \text{sqrt}(dx^2 + dy^2)$	$\sum_{i=1}^{n} \text{sqrt}(\Delta x^2 + \Delta y^2)$

Figure 6: Green's Theorem and discretization

- **Output**

The quantities calculated using the formulas described above are returned to the parent function using arrays.

The following arrays return the corresponding entities.

1. GEOM- Area, perimeter and the coordinates of centroid
2. INER- Moment of inertias about the given axes
3. CPMO- Centroidal principal moment of inertias and the angle they make with the given axes.

Now that we have calculated all the relevant values, we use the following formula to find the value of stress at the point desired.

$$\sigma_{zz} = \frac{M_X Y}{I_X} - \frac{M_Y X}{I_Y}$$

X and Y are replaced with coordinates of the point in consideration and the resulting value is returened as output.

About the GUI[10]

Matlab GUIDE has been used to design the user interface of the software. The layout was designed using various design tools. Coding was done to define the functions of various buttons and to plot the relevant graphs. The major part was connecting the inputs entered by the user to the actual program and then displaying the output generated by the code in an easily interpretable form to the user.

Some common shapes have been predefined in the user interface for quick reference. To perform computations on these structures, the user is asked to enter some defining parameters for the shape. These parameters are then converted into coordinates of the shape and given to the code. This is also used to plot the shape on the graph.

Surf, quiver3 are used for generating 3D plots of the stress acting on the cross section. Every point on the cross section is associated with a corresponding vector indicating the direction and magnitude of stress. The shape is discretized with 0.5mm thickness and the Inpolygon function is used to check if the point lies with the surface. If it does, the code is used to generate the corresponding stress value.

User Manual for the Software

This section is user's manual to the software.

- **Step 1: Selecting the geometry of cross section**

Using the drop-down menu, select the cross section you want to work with. Some common cross sections have been shown.

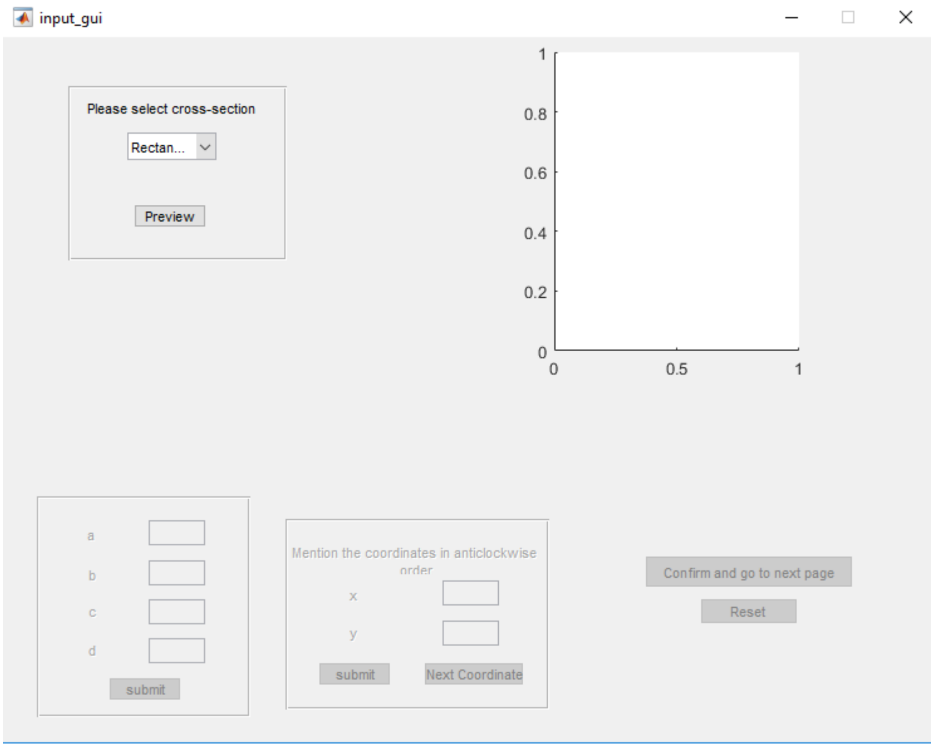


Figure 7 User Manual-Step 1

- **Step 2: Defining parameters**

After selecting the shape, the graph displays your chosen geometry. Press 'Press to provide Input' button to define the parameters of the shape.

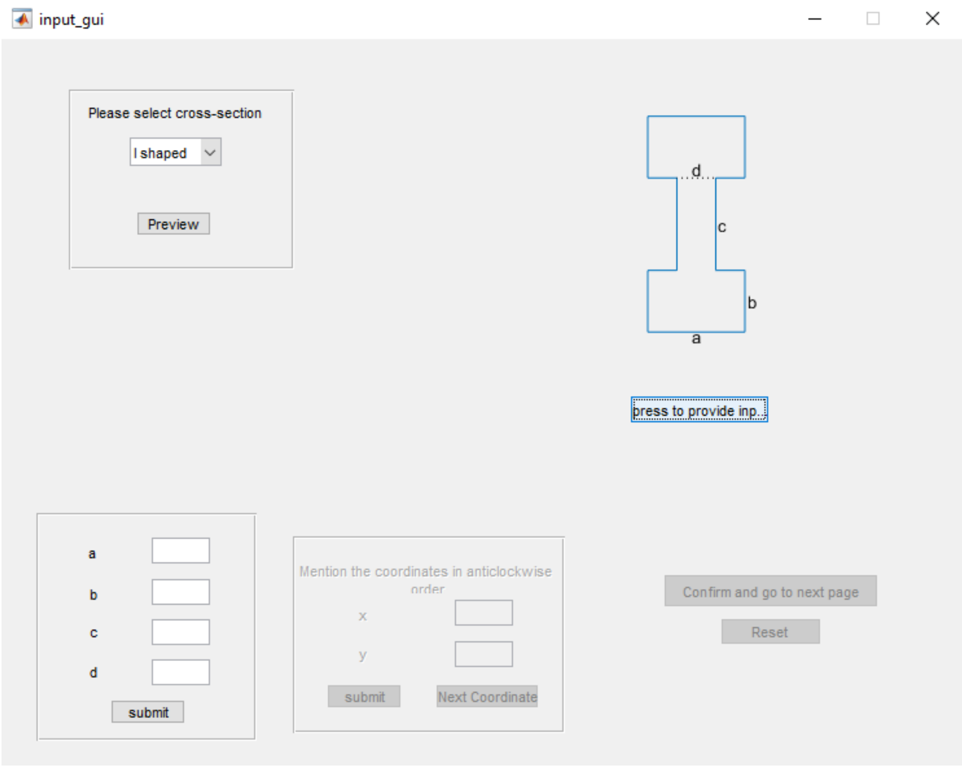


Figure 8 User Manual-Step 2

The parameters have been defined in the figure. Insert the desired values in the corresponding text boxes.

For arbitrary cross section, you have to input end points of the shape that you are working with in anticlockwise sense. Once you have entered all the points and completed a closed loop, press ‘Submit’ button.

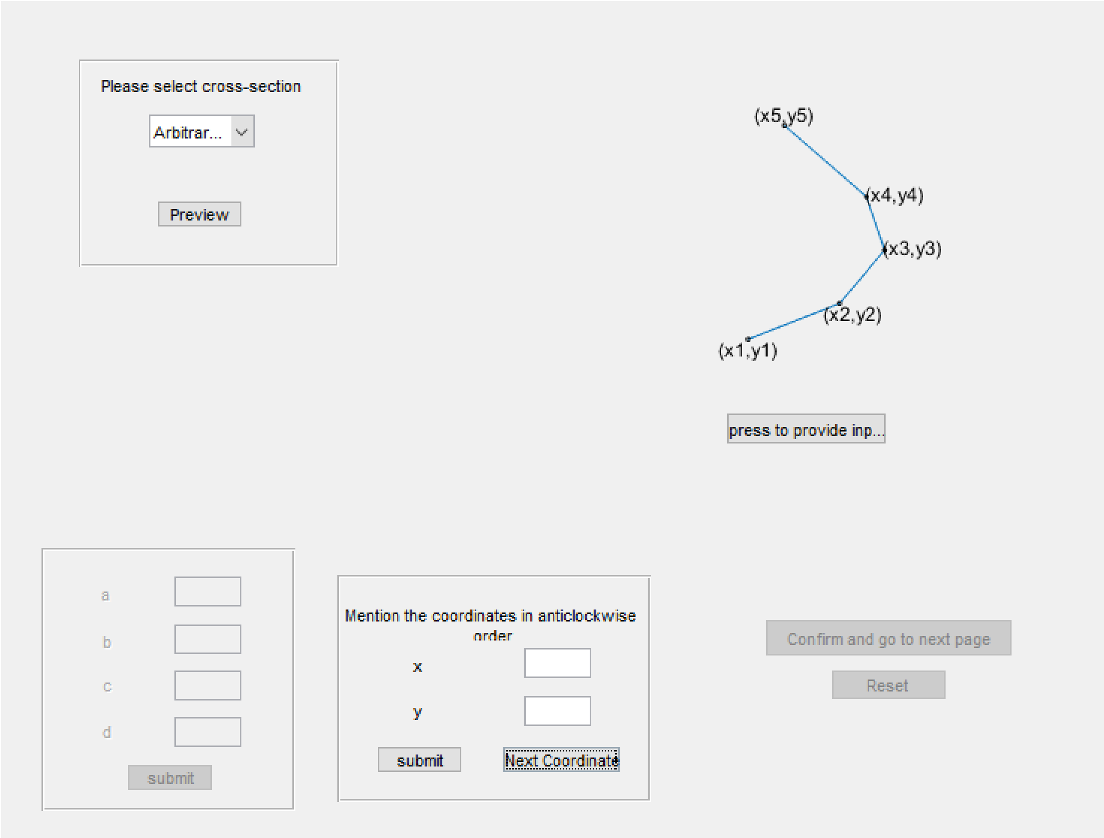


Figure 9 User Manual-Step 2 (Arbitrary)

After entering all the parameters and point, press ‘Confirm and go to next page’ button

- **Step 3: Defining parameter**

On the next page, you will see the geometry of your shape along with the direction of principal axes. Click on‘Click to input moment vales’ button to enter the values of M_x and M_y.

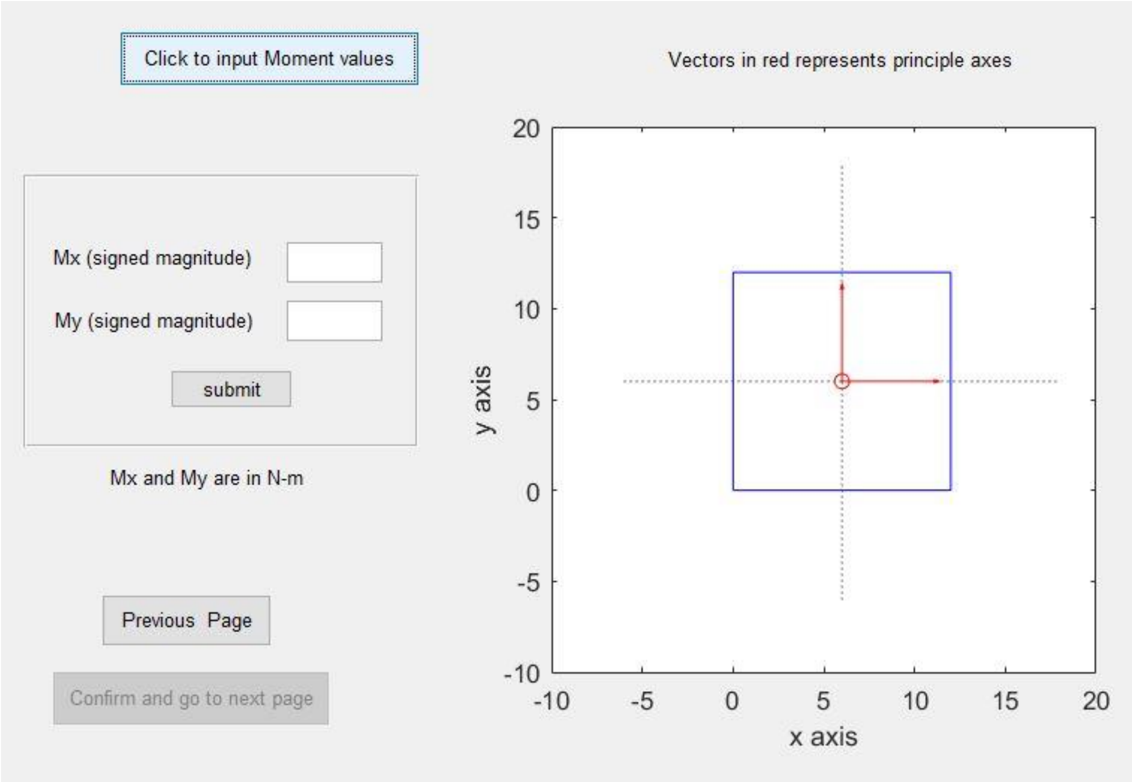


Figure 10 User Manual-Step 3

Press ‘Submit’ button and then ‘Confirm and go to next page’ button.

- **Step 4: Obtaining output**

Press ’Click to proceed’ button. Now you can choose to get the stress at a particular point by entering its coordinates or view the entire stress distribution in 3D

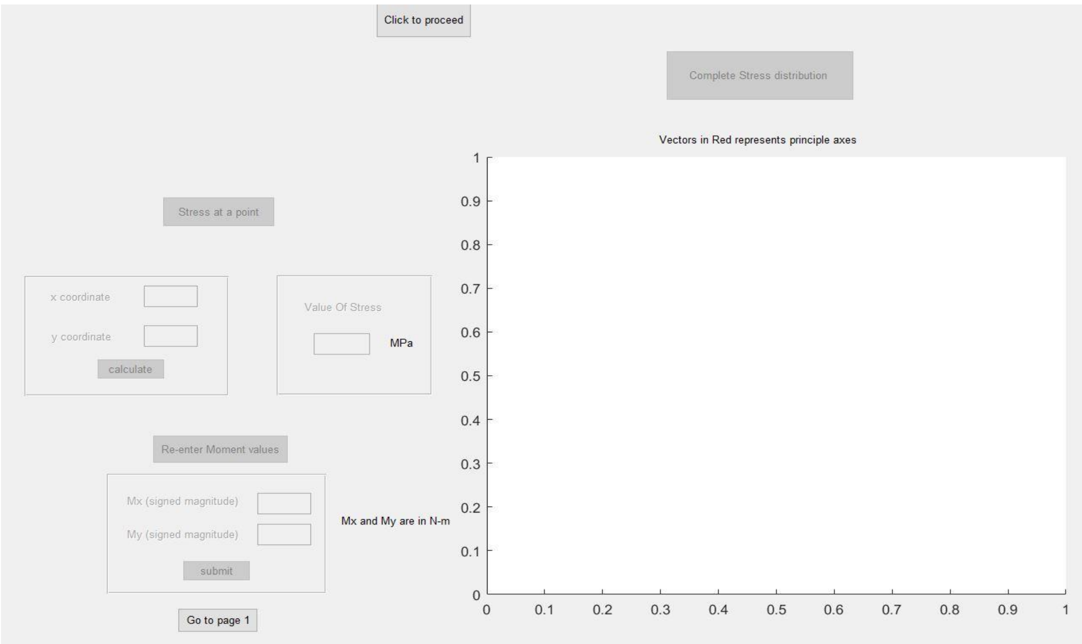


Figure 11 User Manual-Step 4.1

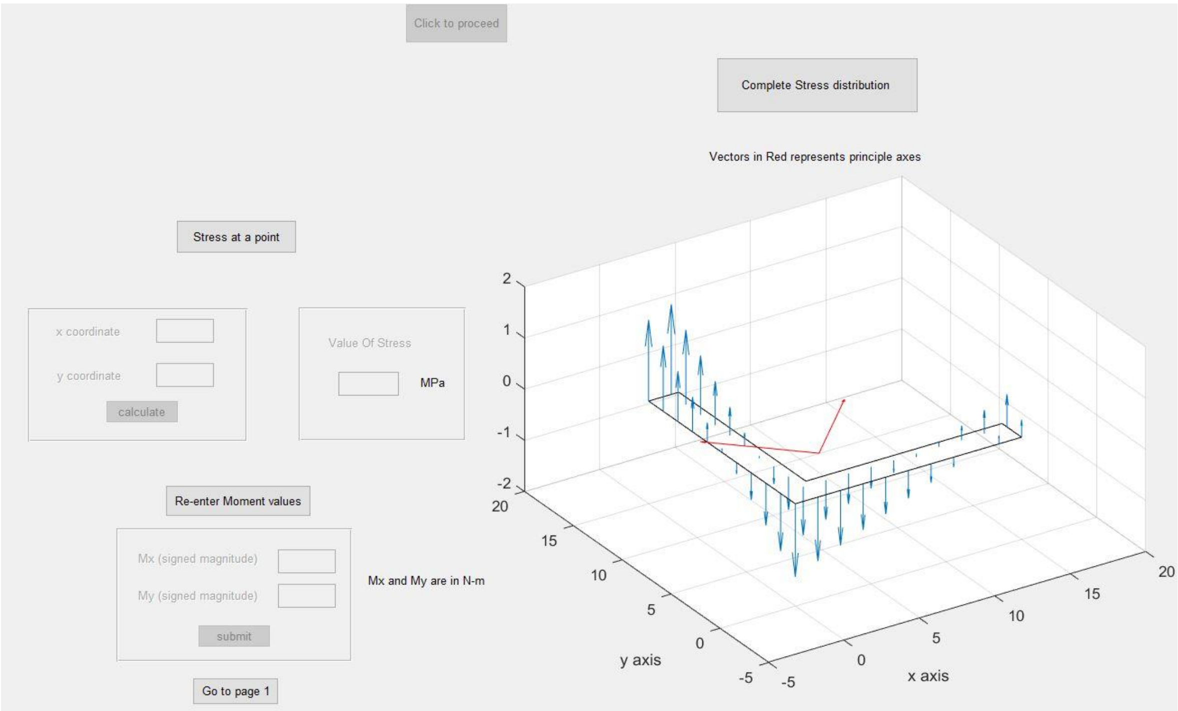


Figure 12 User Manual-Step 4.2

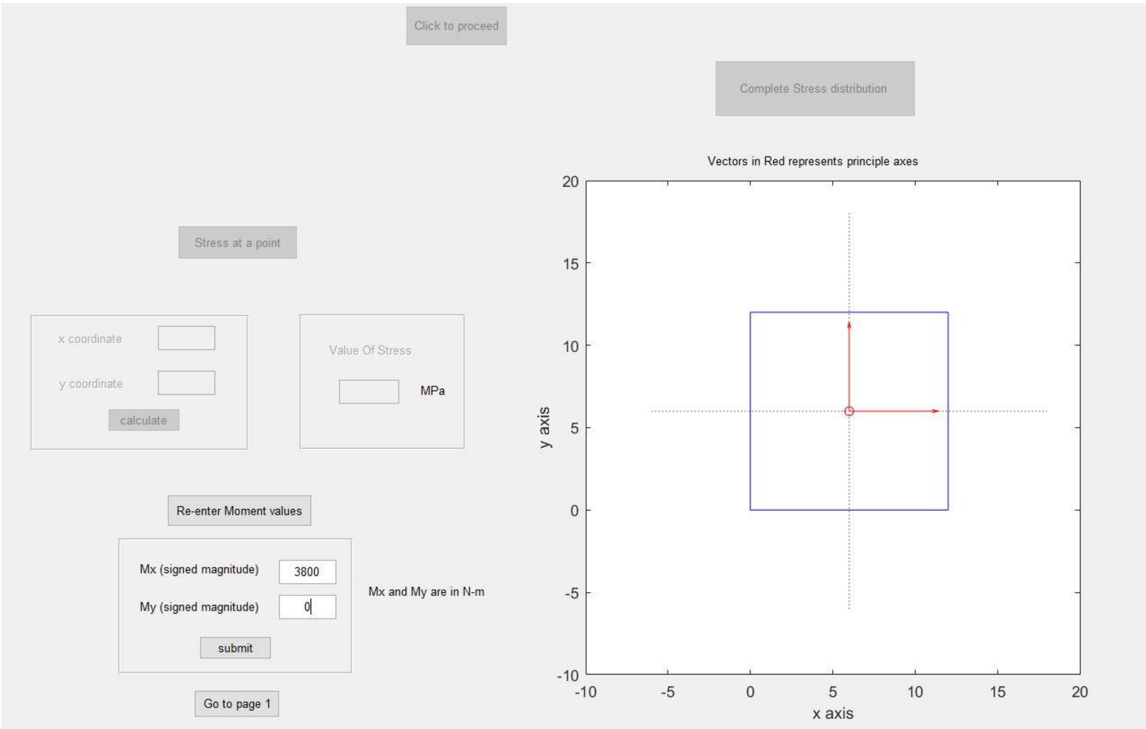


Figure 13 User Manual-Step 4.3

Validation and test cases:

For different cross section cases are divided in two parts.

1. Symmetrical with moment in 1 direction
2. Symmetrical with moment in 2 directions

Symmetrical Bending (Moment acting in 1 direction)

For all cases, axis of beam is taken in z direction. Inputs given to code are coordinates cross sectional points, x and y moment in same reference frame and coordinate of point where we want to find stress. In case of symmetrical bending, moment is applied along x direction only. Given moment is taken acting about centroid of cross section parallel to x axis. And point provided to calculate stress at that point is taken in same reference in which coordinates are given to define geometry.

Calculated and obtained values (from code) of stress for different type of cross section are given below. Inputted coordinates of cross section are in mm and x and y moments are in N.m. Obtained output will be in MPa.

1. Square cross section

Inputs:

Coordinates:

```
x = [ 20  120  120  20];
y = [ 20  20  120  120];
```

Moment:

```
M = [ 10000  0];
```

Beam has below square cross section. Here horizontal axis represents x axis and vertical axis represents y axis.

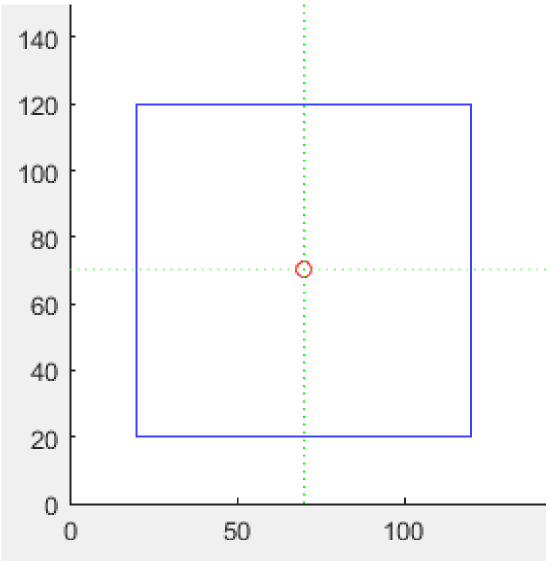


Figure 14 Symmetrical square cross section

`

Stresses calculated for given moment for different points are given below.

Moment applied (kN.m)	Point where moment has to be calculated		Calculated stress (MPa)	
	X component(mm)	Y component(mm)	From code	On paper
10	70	20	-60.0000	-60
10	70	40	-36.0000	-36
10	70	70	0.0000	0
10	70	90	24.0000	24
10	70	120	60.0000	60
10	50	50	-24.0000	-24
10	90	50	-24.0000	-24
10	50	70	0	0
10	90	70	0	0

2. Rectangular cross section

Inputs:

Coordinates: dimensions are in mm.

```
x = [ 20   80   80   20];
y = [ 20   20   120   120];
```

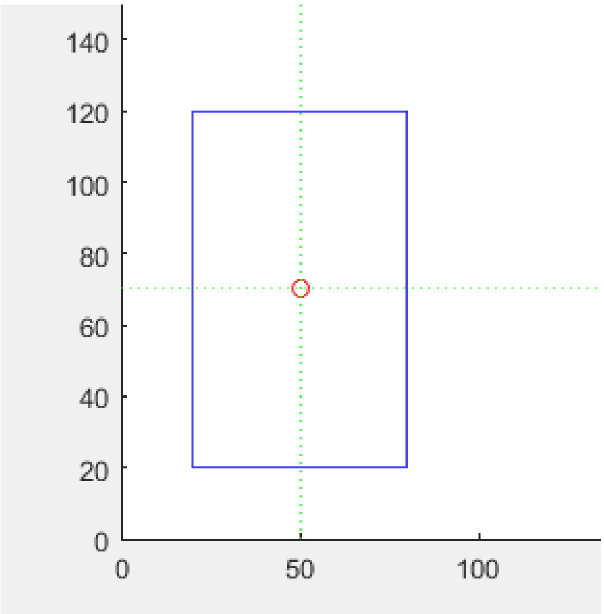


Figure 15 Symmetrical rectangular cross section

Stresses calculated for given moment for different points are given below

Moment applied (kN.m)	Point where moment has to be calculated		Calculated stress (MPa)	
	X component(mm)	Y component(mm)	From code	On paper
8	50	20	-80.0000	-80
8	50	40	-48.0000	-48
8	50	70	0.0000	0
8	50	90	32.0000	32
8	50	120	80.0000	80
12	50	50	-48.0000	-48
12	90	50	-48.0000	-48
12	50	70	0	0
12	90	70	0	0

3. Triangular cross section

Inputs:

Coordinates: dimensions are in mm.

```
x = [ 30   90   60];
y = [ 10   10   110];
```

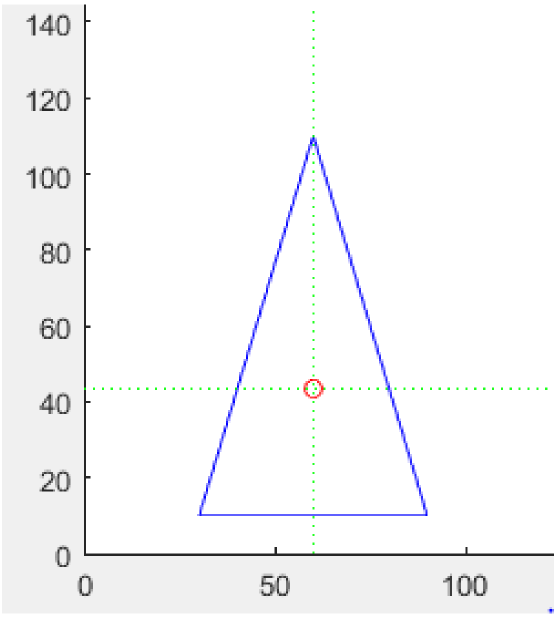


Figure 16 Symmetrical triangular cross section

Stresses calculated for given moment for different points are given below.

Moment applied (kN.m)	Point where moment has to be calculated		Calculated stress (MPa)	
	X component(mm)	Y component(mm)	From code	On paper
6	60	10	-120.0000	119.98
6	60	30	-48.0000	47.98
6	60	60	60.0000	60
6	60	90	168.0000	168
6	60	110	240.0000	240
4	50	30	-32	-31.98
4	70	30	-32	-31.98
4	50	70	64	63.9
4	70	70	64	63.9

4. I type cross section

Inputs:

Coordinates: dimensions are in mm.

```
x = [ 20  50  50  40  40  60  60  10  10  30  30  20];
y = [ 10  10  20  20  60  60  70  70  60  60  20  20];
```

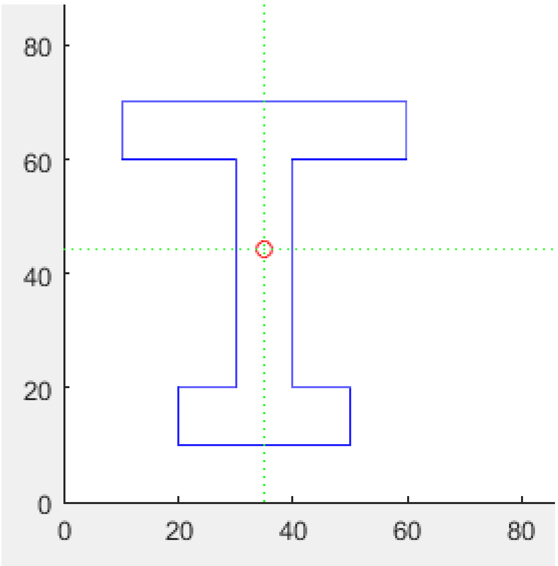


Figure 17 Symmetrical I- cross section

Stresses calculated for given moment for different points are given below.

Moment applied (kN.m)	Point where moment has to be calculated		Calculated stress (MPa)	
	X component(mm)	Y component(mm)	From code	On paper
2.5	35	10	-158.4235	-154.8
2.5	35	20	-112.0556	-108.2
2.5	35	40	-19.3199	-15.44
2.5	35	60	73.4158	77.31
2.5	35	70	119.7836	122.69
3	20	62	99.2272	86.59
3	50	62	99.2272	86.59
3	32	25	-106.6461	-85.01
3	48	25	-106.6461	-85.01

5. T type cross section

Inputs:

Coordinates: dimensions are in mm.

```
x = [ 75  125  125  200  200 0  0  75];
y = [ 0  0  200  200 250  250  200  200];
```

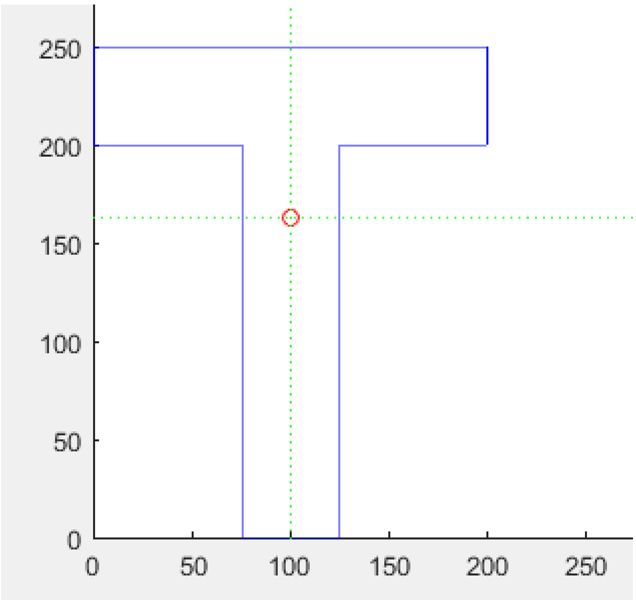


Figure 18 Symmetrical T-cross section

Stresses calculated for given moment for different points are given below.

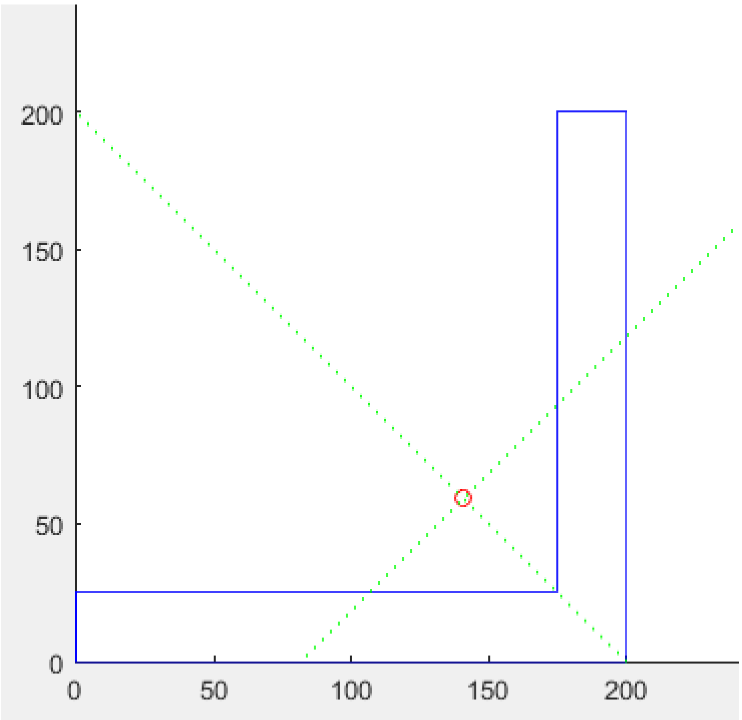
Moment applied (kN.m)	Point where moment has to be calculated		Calculated stress (MPa)	
	X component(mm)	Y component(mm)	From code	On paper
10.5	100	0	-15.0275	-15.027
10.5	100	100	-5.7798	-5.779
10.5	100	175	1.1560	1.155
10.5	100	200	3.4679	3.467
10.5	100	250	8.0917	8.0198
13	85	50	-12.8807	-12.88
13	115	50	-12.8807	-12.88
13	50	230	7.7284	7.73
13	150	230	7.7284	7.73

6. L type cross section

Inputs:

Coordinates: dimensions are in mm.

```
x = [ 0  200  200  175  175  0];
y = [ 0  0  200  200  25  25];
```



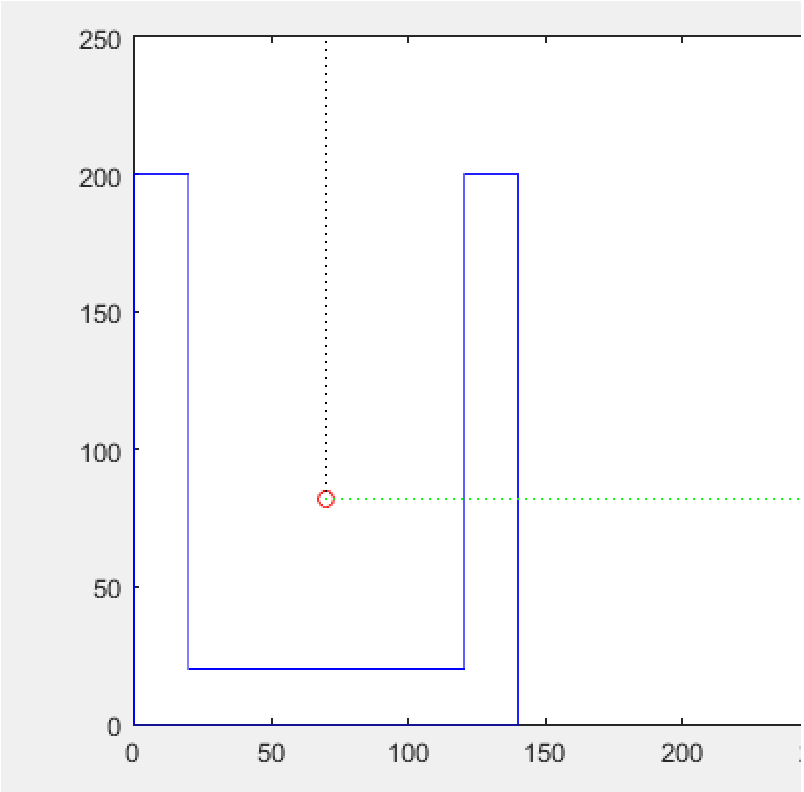
Stresses calculated for given moment for different points are given below.

Moment applied (kN.m)	Point where moment has to be calculated		Calculated stress (MPa)	
	X component(mm)	Y component(mm)	From code	On paper
10	200	0	-412.5	-413
10	200	118.34	107.262	107.25
10	81	0	-105.513	-107.25
10	0	0	103.46	104.079

7. U type cross section

Input:

```
x = [ 0  140  140  120  120  20  20  0];
y = [ 0  0  200  200  20  20  200  200];
```



Moment applied (kN.m)	Point where moment has to be calculated		Calculated stress (MPa)	
	X component(mm)	Y component(mm)	From code	On paper
12	10	0	-24.7901	-24.78
12	10	50	-9.6742	-9.67
12	10	100	5.4417	5.44
12	10	150	20.5576	20.55
12	10	200	35.6735	35.68
13	0	10	-23.5808	-23.58
13	50	10	-23.5808	-23.58
13	100	10	-23.5808	-23.58
13	140	10	-23.5808	-23.58

8. Circle, semicircle

Code is not able to take input whose cross section has not had finite coordinates. So for geometries like circle and semicircle, code is not working.

Symmetrical Bending (Moment acting in 2 directions)

For given cross section, there is moment about both x and y direction.

1. Square cross section

Inputs:

Coordinates: dimensions are in mm.

```
x = [ 20   120   120   20];
y = [ 20   20   120   120];
```

Beam has below square cross section. Here horizontal axis represents x axis and vertical axis represents y axis.

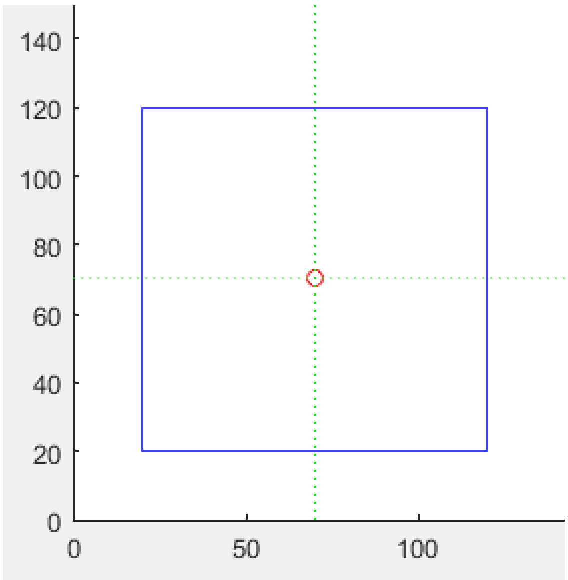


Figure 19 Unsymmetrical square cross section

`

Stresses calculated for given moment for different points are given below

X direction moment (kN.m)	Y direction moment (kN.m)	Point where moment has to be calculated		Calculated stress (MPa)	
		X component(mm)	Y component(mm)	From code	On paper
5	5	40	40	0.0000	0
5	5	60	60	0.0000	0
5	5	70	70	0.0000	0
5	5	100	100	0.0000	0
5	5	40	100	36	36
5	5	60	80	12	12
5	5	90	50	-24	-24
5	5	100	40	-36	-36
10	5	50	30	-36.0000	-36
10	5	60	50	-18.0000	-18
10	5	70	70	0	0
10	5	90	110	36.0000	36
10	5	30	70	24.0000	24
10	5	60	70	6.0000	6
10	5	80	70	-6.0000	-6
10	5	110	70	-24.0000	-24

2. Rectangular cross section

Inputs:

Coordinates: dimensions are in mm.

```
x = [ 20   80   80   20];
y = [ 20   20   120   120];
```

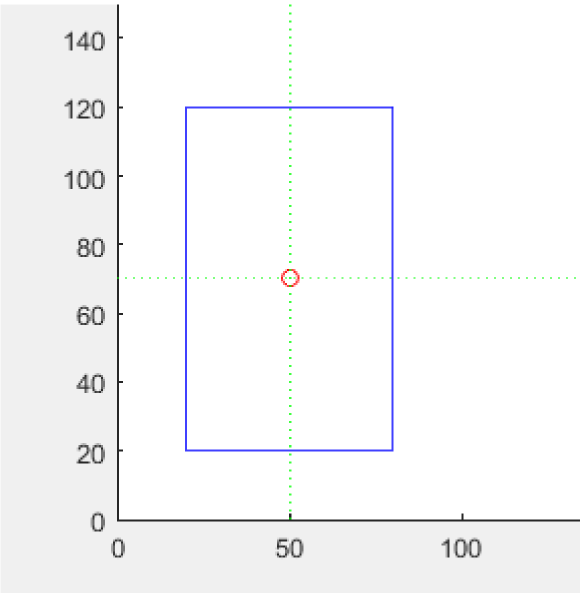


Figure 20 Unsymmetrical rectangular cross section

Stresses calculated for given moment for different points are given below.

X direction moment (kN.m)	Y direction moment (kN.m)	Point where moment has to be calculated		Calculated stress (MPa)	
		X component(mm)	Y component(mm)	From code	On paper
8	5	50	30	-64	-64
8	5	50	50	-32	-32
8	5	50	70	0	0
8	5	50	90	32	32
8	5	50	110	64	64
5	9	40	20	0	0
5	9	50	70	0	0
5	9	60	120	0	0
10	5	60	70	-27.7778	-27.77

3. Triangular cross section

Inputs:

Coordinates: dimensions are in mm.

```
x = [ 30   90   60];
y = [ 10   10   110];
```

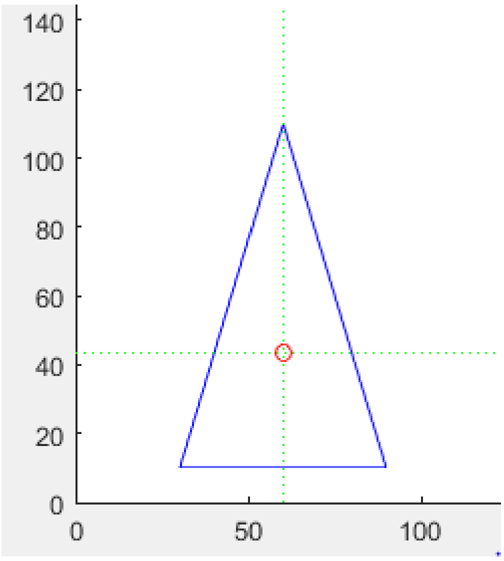


Figure 21 Unsymmetrical triangular cross section

Stresses calculated for given moment for different points are given below.

X direction moment (kN.m)	Y direction moment (kN.m)	Point where moment has to be calculated		Calculated stress (MPa)	
		X component(mm)	Y component(mm)	From code	On paper
8	6	60	10	-160.0000	-160
8	6	60	30	-64.0000	-64
8	6	60	60	80.0000	80
8	6	60	90	224.0000	224
8	6	60	110	320.0000	320
6	4	50	20	4.8889	4.9
6	4	57.5	40	10.2222	10.012
6	4	65	60	15.5556	15
6	4	70	70	7.1111	7

4. I type cross section

Inputs:

Coordinates: dimensions are in mm.

```
x = [ 20   50   50   40   40   60   60   10   10   30   30   20];
y = [ 10   10   20   20   60   60   70   70   60   60   20   20];
```

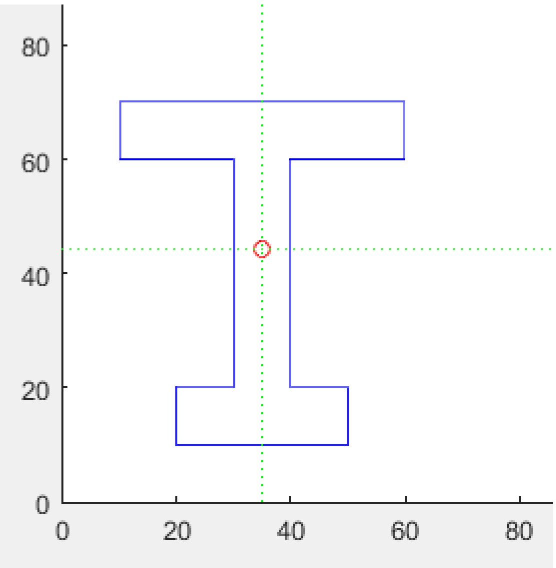


Figure 22 Unsymmetrical I-cross section

Stresses calculated for given moment for different points are given below.

X direction moment (kN.m)	Y direction moment (kN.m)	Point where moment has to be calculated		Calculated stress (MPa)	
		X component(mm)	Y component(mm)	From code	On paper
3	2.5	35	10	-190.1082	-189.88
3	2.5	35	20	-134.4668	-134
3	2.5	35	40	-23.1839	-23
3	2.5	35	60	88.0989	88
3	2.5	35	70	143.7403	143
3	2	40	10	-267.0313	-266.7
3	2	40	20	-211.3898	-211
3	2	40	60	11.1758	11
3	2	40	70	66.8173	66
1.5	3	20	62	395.7674	395
1.5	3	50	62	-296.5402	296
1.5	3	32	25	15.9077	15
1.5	3	48	25	-353.3230	353

5. T type cross section

Inputs:

Coordinates: dimensions are in mm.

```
x = [ 75  125  125  200  200 0  0  75];
y = [ 0  0  200  200 250  250  200  200];
```

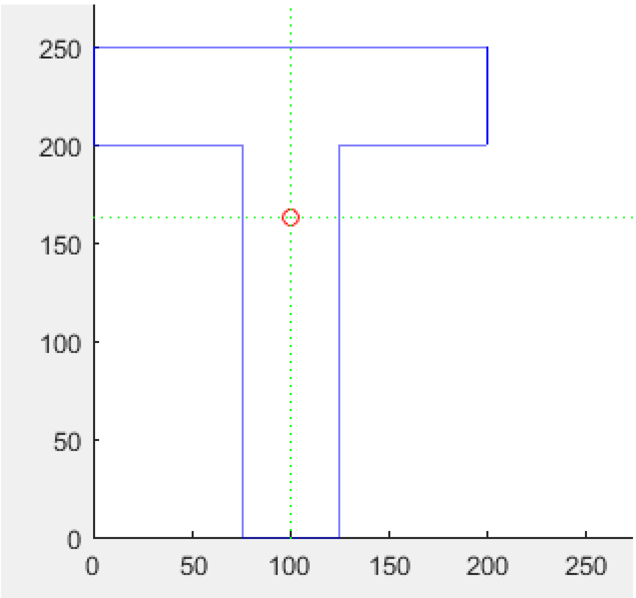


Figure 23 Unsymmetrical T-cross section

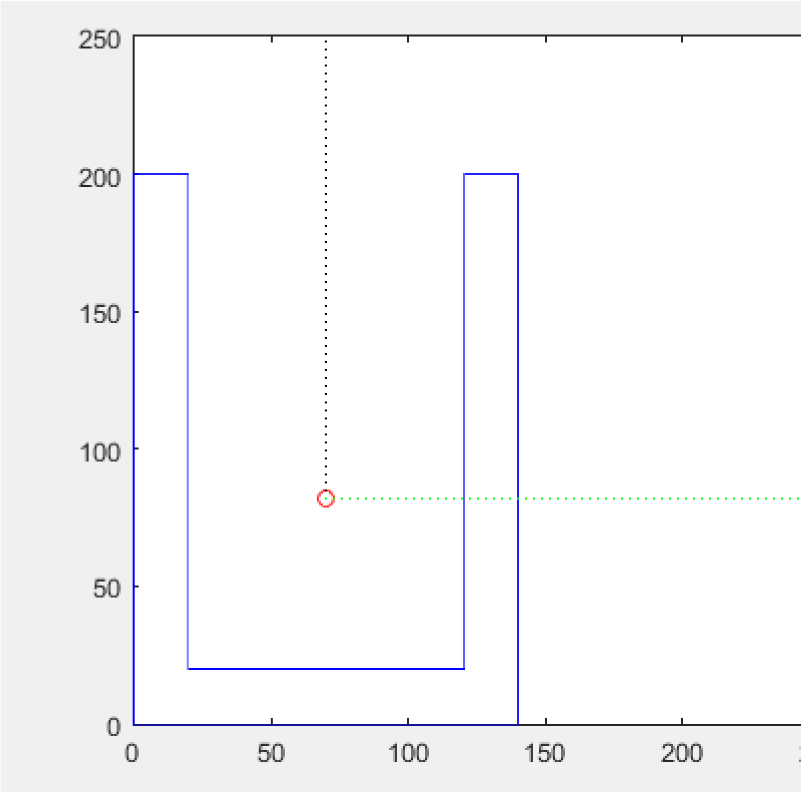
Stresses calculated for given moment for different points are given below.

X direction moment (kN.m)	Y direction moment (kN.m)	Point where moment has to be calculated		Calculated stress (MPa)	
		X component(mm)	Y component(mm)	From code	On paper
7.5	9.5	100	0	-10.7339	-10.734
7.5	9.5	100	100	-4.1284	-4.128
7.5	9.5	100	175	0.8257	0.825
7.5	9.5	100	200	2.4771	2.477
7.5	9.5	100	250	5.7798	5.779
10	6	10	250	22.9535	22.955
10	6	50	250	16.1770	16.175
10	6	150	250	-0.7642	-0.775
10	6	190	250	-7.5406	-7.555
8	13	85	50	-2.4207	-2.426
8	13	115	50	-13.4325	-13.436
8	13	50	230	23.1089	23.108
8	13	150	230	-13.5970	-13.59

6. U section

Input:

```
x = [ 0   140   140   120   120   20   20   0];
y = [ 0   0   200   200   20   20   200   200];
```



X direction moment (kN.m)	Y direction moment (kN.m)	Point where moment has to be calculated		Calculated stress (MPa)	
		X component(mm)	Y component(mm)	From code	On paper
8	12.5	10	0	7.8768	7.90
8	12.5	10	50	17.9540	17.97
8	12.5	10	100	28.0313	28.05
8	12.5	10	150	38.1085	38.13
8	12.5	10	200	48.1858	48.21
8	12.5	130	0	-40.9302	-40.95
8	12.5	130	50	-30.8529	-30.87
8	12.5	130	100	-20.7757	-20.80
8	12.5	130	150	-10.6984	-10.70
8	12.5	130	200	-0.6211	-0.64
5	8	0	10	9.1517	9.17
5	8	50	10	-3.8635	-3.85
5	8	100	10	-16.8786	-16.88
5	8	140	10	-27.2908	-27.31

Limitations of the software

1. The code is applicable only when the plane of load passes through the Shear centre of the cross section. Moreover, it is applicable only in the Pure Bending scenario.
2. The code is applicable for any arbitrary polygonal cross section however it is not applicable for any non-polygonal cross section.
3. The code requires the Moment Vector acting on the arbitrary cross section to determine the stresses acting on the cross section.
4. In order to make the code work for arbitrary polygonal cross section we compromised with the easy inputs like web length, flange width for I section and went for coordinates of the cross section as an input.
5. Neutral Axis is not computed in the code as it is left to the user to visualize the neutral axis from the visualization of the stress distribution on the cross section.
6. The software can't calculate stresses for unsymmetrical cases. Only the problems in which shear centre coincides with the centroidal axis is considered.
7. For arbitrary cross-sections the code sometimes assigns the direction of principal axes unproperly leading to false results as the direction of the moment vector is distorted.
8. The code is not applicable for beams with curved cross sections
9. GUI is multi-paged which makes the comparison between cases difficult

References:

[1] “Blazingsaddles” [Online]. Available: https://www.blazingsaddles.com/. [Accessed: 11-Nov-2017].

[2] “Neutral Axis Lies Midway between the Top and Bottom of Beams with Vertical Symmetry | Structural Systems | Pinterest” [Online]. Available: https://www.pinterest.com/pin/272819689902442982/. [Accessed: 07-Nov-2017].

[3] “Moment of Inertia” [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/mi2.html. [Accessed: 07-Nov-2017].

[4] “Moments of Inertia” [Online]. Available: https://web2.ph.utexas.edu/~coker2/index.files/RI.htm. [Accessed: 07-Nov-2017].

[5] “Chapter 05 - Stresses in Beams | Strength of Materials Review” [Online]. Available: https://www.mathalino.com/reviewer/mechanics-and-strength-of-materials/chapter-5-stresses-in-beams. [Accessed: 07-Nov-2017].

[6] “Flexure Formula | Strength of Materials Review” [Online]. Available: https://www.mathalino.com/reviewer/mechanics-and-strength-of-materials/flexure-formula. [Accessed: 07-Nov-2017].

[7] Archer, R., Cook, N., Crandall, S., Dahl, N., Lardner, T., McClintock, F., Rabinowicz, E., and ReichenbachGeorge, *An Introduction to the Mechancs of Solids*.

[8] “Polygeom.m - File Exchange - MATLAB Central” [Online]. Available: https://in.mathworks.com/matlabcentral/fileexchange/319-polygeom-m. [Accessed: 11-Nov-2017].

[9] “No Title” [Online]. Available: www.mne.psu.edu/sommer/me481/notes_07_02.docx. [Accessed: 11-Nov-2017].

[10] “MATLAB Answers - MATLAB Central” [Online]. Available: https://in.mathworks.com/matlabcentral/answers. [Accessed: 12-Nov-2017].

List of Figures:

Figure 1: Compression and Tension in a Beam[2] 3
Figure 2: Moment of Inertia of some common shapes[4]..3
Figure 3 Bending in a beam 4
Figure 4 Longitudinal cross section of a beam 4
Figure 5 Derivation of stresses in beams 5
Figure 6: Green's Theorem and discretization 8
Figure 7 User Manual-Step 1 10
Figure 8 User Manual-Step 2 10
Figure 9 User Manual-Step 2 (Arbitrary) 11
Figure 10 User Manual-Step 3 12
Figure 11 User Manual-Step 4.1 12
Figure 12 User Manual-Step 4.2 13
Figure 13 User Manual-Step 4.3 13
Figure 14 Square cross section Symmetrical 15
Figure 15 Symmetrical rectangular cross section 16
Figure 16 Symmetrical triangular cross section...17
Figure 17 Symmetrical I- cross section 18
Figure 18 Symmetrical T-cross section 19
Figure 19 Unsymmetrical square cross section 22
Figure 20 Unsymmetrical rectangular cross section...24
Figure 21 Unsymmetrical triangular cross section 25
Figure 22 Unsymmetrical I-cross section 26
Figure 23 Unsymmetrical T-cross section 27

Appendix

Code for Test Function

Test function takes the input from the user and displays the output. It is the main function of the software.

```
% Polygonal Cross Section; Symmetric as well as Unsymmetric Pure Bending
% Scenario with Moment vector given at the cross section with phi as the
% angle between the plane of load and x-z plane; phi is taken positive
% counter-clockwise
clear
% constants
d2r = pi / 180;

% 3x5 test rectangle with long axis at 30 degrees
% area=15, x_cen=3.415, y_cen=6.549, perimeter=16
% I1=11.249, I2=31.247, J=42.496
x = [ 0  140  140  120  120  20  20  0]; % Enter X and Y coordinates of the
y = [ 0  0  200  200  20  20  200  200]; % points on cross section as
matrices


M=[5000,8000];    % Enter Moment Vector acting on the the centroid of the
cross secrion as a vector
theta=atan(M(2)/M(1));
xx=140;           % Enter the x and y coordinate of the point at which you
want to compute the stress wrt your defined X-Y coordinate axis
yy=10;

[geom,iner,cpmo,eig_vec, xm, ym ]=polygeom(x,y);
%applying the function defined by the name polygeom in the polygeom.m file.
%Here x and y are the vectors representing the x and y coordinates of the
%points on the arbitrary polygonal cross section. This points are necessary
%to feed the algorithm, information about the geometry of the cross
%section.

A=geom(1);
X_C=geom(2);
Y_C=geom(3);
P=geom(4);
%geom = [ area   X_cen  Y_cen  perimeter ]
%iner = [ Ixx     Iyy     Ixy     Iuu     Ivv     Iuv ]
%u,v are centroidal axes parallel to x,y axes.
%cpmo= [ I1      ang1    I2      ang2    J ]
%I1,I2 are centroidal principal moments about axes
%at angles ang1,ang2.
%ang1 and ang2 are in radians.
%J is centroidal polar moment.
%J = I1 + I2 = Iuu + Ivv

I_xx=iner(1);
I_yy=iner(2);
I_xy=iner(3);
I_uu=iner(4);
```

```
I_vv=iner(5);
I_uv=iner(6);

I1=cpmo(1);
ang1=cpmo(2);
I2=cpmo(3);
ang2=cpmo(4);
J=cpmo(5);

%plot cross section%
xplot = x( [ 1:end 1] );
yplot = y( [ 1:end 1] );
rad = 350;
x1 = [ X_C-rad*cos(ang1)  X_C+rad*cos(ang1) ];
y1 = [ Y_C-rad*sin(ang1)  Y_C+rad*sin(ang1) ];
x2 = [ X_C-rad*cos(ang2)  X_C+rad*cos(ang2) ];
y2 = [ Y_C-rad*sin(ang2)  Y_C+rad*sin(ang2) ];
plot( xplot,yplot,'b', X_C,Y_C,'ro', ...
      x1,y1,'g:', x2,y2,'g:'  )
axis( [ 0  rad  0  rad ] )
axis square

% Once we know the Principal Moments of Inertia what we can do is to
% project the Moment Vector acting on a given cross section along the
% direction of the principal axis.
% Thereafter calculating the norms of the respective projected vectors,
% we get M_1 and M_2.
% Now we can just plug in the above calculated quantities in the formula to
% determine stress at a particular point on the cross section.
% The necessary formula is mentioned as Equation (7.4) in Chapter 7 Bending
% of Straight Beams, Page No. 268 in Advanced Mechanics of Materials,
% 6th Edition Arthur P. Boresi, Richard J. Schmidt; ISBN: 978-0-471-43881-6
% if ang1*theta<0
%     gamma1=ang1+theta;
% end
% if ang1*theta>0
%     gamma1=ang1-theta;
% end
%
% if ang2*theta<0
%     gamma2=ang2+theta;
% end
% if ang2*theta>0
%     gamma2=ang2-theta;
% end
%
% if theta ==0
%     gamma1=ang1;
%     gamma2=ang2;
% end

M_11=proj(M,[eig_vec(1,1), eig_vec(2,1)]);
M_22=proj(M,[eig_vec(1,2), eig_vec(2,2)]);
M_1=norm(M_11);
M_2=norm(M_22);
M_111=norm(M)*cos(theta+ang1);
M_222=norm(M)*cos((d2r*90)+ang1+theta);
```

```
% shift the coordinates from the specified x-y coordinate frame defined by
% the user when she/he was giving the coordinates of the cross section as
% input to the centroidal coordinate frame and then to the Principal Axis
% Coordinate Frame by transforming the coordinates by angle ang1.
% We are doing this as X and Y coordinates of the point at which the user
% requires the Stress is wrt the coordinate axis defined by the user
% whereas in the Equation (7.4) in Chapter 7 Bending
% of Straight Beams, Page No. 268 in Advanced Mechanics of Materials,
% 6th Edition Arthur P. Boresi, Richard J. Schmidt; ISBN: 978-0-471-43881-6
% x and y coordinates (in the formula) is wrt thw Principal Axis.
xx=xx-X_C;
yy=yy-Y_C;
XT=(xx*cos(ang1))+(yy*sin(ang1));
YT=(-1*xx*sin(ang1))+(yy*cos(ang1));
if ang1==0 || ang2==0
    S=-1*((((M_1*xx)/I1)-((M_2*yy)/I2))*1000);
else
    S=((((M_111*YT)/I1)-((M_222*XT)/I2))*10000);
%      phi=input('Enter phi (in degrees): ');
%      phi=d2r*phi;
%      Mx=norm(M)*sin(phi);
%      ta=(I_uv-(I_uu*cot(phi)))/(I_vv-(I_uv*cot(phi)));
%      S=(Mx*(yy-(xx*ta)))/I_uu-(I_uv*ta);
end
```

Code for Polygeon Function

The Polygeom function calculates various quantities such as moment of inertias, coordinates of centroids etc.

```
function [ geom, iner, cpmo ,eig_vec, xm, ym] = polygeom( x, y )
%POLYGEOM Geometry of a planar polygon
%
%   POLYGEOM( X, Y ) returns area, X centroid,
%   Y centroid and perimeter for the planar polygon
%   specified by vertices in vectors X and Y.
%
%   [ GEOM, INER, CPMO ] = POLYGEOM( X, Y ) returns
%   area, centroid, perimeter and area moments of
%   inertia for the polygon.
%   GEOM = [ area   X_cen  Y_cen  perimeter ]
%   INER = [ Ixx    Iyy    Ixy    Iuu    Ivv    Iuv ]
%     u,v are centroidal axes parallel to x,y axes.
%   CPMO = [ I1     ang1   I2     ang2   J ]
%     I1,I2 are centroidal principal moments about axes
%         at angles ang1,ang2.
%     ang1 and ang2 are in radians.
%     J is centroidal polar moment.  J = I1 + I2 = Iuu + Ivv

% H.J. Sommer III - 16.12.09 - tested under MATLAB v9.0
%
% sample data
% x = [ 2.000  0.500  4.830  6.330 ]';
% y = [ 4.000  6.598  9.098  6.500 ]';
```

```
% 3x5 test rectangle with long axis at 30 degrees
% area=15, x_cen=3.415, y_cen=6.549, perimeter=16
% Ixx=659.561, Iyy=201.173, Ixy=344.117
% Iuu=16.249, Ivv=26.247, Iuv=8.660
% I1=11.249, ang1=30deg, I2=31.247, ang2=120deg, J=42.496
%
% H.J. Sommer III, Ph.D., Professor of Mechanical Engineering, 337 Leonhard
Bldg
% The Pennsylvania State University, University Park, PA  16802
% (814)863-8997  FAX (814)865-9693  hjs1-at-psu.edu
www.mne.psu.edu/sommer/

% begin function POLYGEOM

% check if inputs are same size
if ~isequal( size(x), size(y) )
  error( 'X and Y must be the same size');
end

% temporarily shift data to mean of vertices for improved accuracy
xm = mean(x);
ym = mean(y);
x = x - xm;
y = y - ym;
K=[xm  ym];
% summations for CCW boundary
xp = x( [2:end 1] );
yp = y( [2:end 1] );
a = x.*yp - xp.*y;

A = sum( a ) /2;
xc = sum( (x+xp).*a  ) /6/A;
yc = sum( (y+yp).*a  ) /6/A;
Ixx = sum( (y.*y +y.*yp + yp.*yp).*a  ) /12;
Iyy = sum( (x.*x +x.*xp + xp.*xp).*a  ) /12;
Ixy = sum( (x.*yp +2*x.*y +2*xp.*yp + xp.*y).*a  ) /24;

dx = xp - x;
dy = yp - y;
P = sum( sqrt( dx.*dx +dy.*dy ) );

% check for CCW versus CW boundary
if A < 0
  A = -A;
  Ixx = -Ixx;
  Iyy = -Iyy;
  Ixy = -Ixy;
end

% centroidal moments
Iuu = Ixx - A*yc*yc;
Ivv = Iyy - A*xc*xc;
Iuv = Ixy - A*xc*yc;
J = Iuu + Ivv;

% replace mean of vertices
x_cen = xc + xm;
y_cen = yc + ym;
```

```
Ixx = Iuu + A*y_cen*y_cen;
Iyy = Ivv + A*x_cen*x_cen;
Ixy = Iuv + A*x_cen*y_cen;

% principal moments and orientation
I = [ Iuu  -Iuv ;
     -Iuv   Ivv ];
[ eig_vec, eig_val ] = eig(I);
I1 = eig_val(1,1);
I2 = eig_val(2,2);
ang1 = atan2( eig_vec(2,1), eig_vec(1,1) );
ang2 = atan2( eig_vec(2,2), eig_vec(1,2) );

% return values
geom = [ A  x_cen  y_cen  P ];
iner = [ Ixx  Iyy  Ixy  Iuu  Ivv  Iuv];
cpmo = [ I1  ang1  I2  ang2  J ];

% bottom of polygeom
```

Code used for GUI

Code for input gui Function

This is the main program for the GUI

```
function varargout = input_gui(varargin)
% INPUT_GUI MATLAB code for input_gui.fig
%      INPUT_GUI, by itself, creates a new INPUT_GUI or raises the existing
%      singleton*.
%
%      H = INPUT_GUI returns the handle to a new INPUT_GUI or the handle to
%      the existing singleton*.
%
%      INPUT_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in INPUT_GUI.M with the given input arguments.
%
%      INPUT_GUI('Property','Value',...) creates a new INPUT_GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before input_gui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to input_gui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help input_gui

% Last Modified by GUIDE v2.5 12-Nov-2017 22:30:40

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
```

```
                   'gui_OpeningFcn', @input_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @input_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before input_gui is made visible.
function input_gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to input_gui (see VARARGIN)


% Choose default command line output for input_gui

handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

%Generating the intial view for the input window by disabling unnecessary edit,text
fields and pushbuttons
new_handle_array=[handles.text_x_array,handles.text_y_array,handles.text_a,handles.
text_b,handles.text_c,handles.text_d,handles.text_input,];
set(new_handle_array,'Value',[]);
Base_handle_array=[handles.text_I_a, handles.text_I_b, handles.text_I_c,
handles.text_I_d, handles.text_arbit_x...
    ,handles.text_arbit_y, handles.text_arbit_top, handles.edit_I_a,
handles.edit_I_b, handles.edit_I_c, handles.edit_I_d...
    ,handles.edit_arbit_x, handles.edit_arbit_y, handles.pushbutton_I_submit,
handles.pushbutton_arbit_next, handles.pushbutton_arbit_submit...
    ,handles.pushbutton_reset,handles.pushbutton_nextpage];
set(Base_handle_array,'Enable','off');

% UIWAIT makes input_gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = input_gui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes during object creation, after setting all properties.
function cr_sec_panel_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cr_sec_panel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% --- Executes when cr_sec_panel is resized.
function cr_sec_panel_SizeChangedFcn(hObject, eventdata, handles)
% hObject    handle to cr_sec_panel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selection change in pop_crosssectiontype.
function pop_crosssectiontype_Callback(hObject, eventdata, handles)
% hObject    handle to pop_crosssectiontype (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = cellstr(get(hObject,'String')) returns pop_crosssectiontype
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
pop_crosssectiontype


% --- Executes during object creation, after setting all properties.
function pop_crosssectiontype_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pop_crosssectiontype (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton_preview.
function pushbutton_preview_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_preview (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for preview button . It generates the figure showing
%cross-section type
handle_array=[handles.text_I_a, handles.text_I_b, handles.text_I_c,
handles.text_I_d, handles.text_arbit_x...
    ,handles.text_arbit_y, handles.text_arbit_top, handles.edit_I_a,
handles.edit_I_b, handles.edit_I_c, handles.edit_I_d...
    ,handles.edit_arbit_x, handles.edit_arbit_y, handles.pushbutton_I_submit,
handles.pushbutton_arbit_next, handles.pushbutton_arbit_submit];
set(handle_array,'Enable','off');
val2=get(handles.pop_crosssectiontype,'Value');

% plotting the figure showing cross-section type
switch val2
    case 1   % rectangle
        set(handles.text_input,'Value',1);
        a=5;  %width
        b=2;  %height
        x = 4;
        y = 1;
        base_x = [x x+a x+a  x   x];
        base_y = [y  y  y+b y+b y];

        plot(base_x, base_y,'-');
        hold on;
        axis([2 a+6 0 b+3]);
```

```
        %labelling the figure
        eps_x=0.16; % offset in x
        eps_y=0.12; % offset in y
        text(x+a+eps_x+eps_x+eps_x, y+b/2,
'b','HorizontalAlignment','right');%labelling b
        text(x+a/2, y+b+eps_y+eps_y, 'a','HorizontalAlignment','right');%labelling
a
        hold off;
        axis off;


    case 2  % Triangle
        set(handles.text_input,'Value',2);
        a=5;  %width
        b=2;  %height
        x = 4;
        y = 1;
        base_x = [x x+a x+a/2 x ];
        base_y = [y y y+b y];

        plot(base_x, base_y,'-');
        hold on;
        axis([2 a+6 0 b+3]);
        %labelling the figure
        eps_x=0.16; % offset in x
        eps_y=0.12; % offset in y
        plot([x+(a/2) x+(a/2)],[y y+b],'k:');  % dimension b
        text(x+(a/2)+2.4*eps_x, y+b/2, 'b','HorizontalAlignment','right');
%labelling b
        text(x+a/2, y-eps_y, 'a','HorizontalAlignment','center'); %labelling a
        hold off;
        axis off;


    case 3   % I shaped cross-section
        set(handles.text_input,'Value',3);
        a=5;  %width
        b=2;  %height
        c=3;
        d=2;
        e=(a-d)/2;
        x = 4;
        y = 1;
        base_x = [x x+a  x+a  x+a-e  x+a-e   x+a    x+a     x       x    x+e  x+e x
x ];  % x coordinates (anticlockwise)
        base_y = [y  y   y+b  y+b    y+b+c  y+b+c y+b+c+b y+b+c+b y+b+c y+b+c y+b
y+b y ];  % x coordinates (anticlockwise)
        plot(base_x, base_y,'-');
        hold on;
        % Axes properties
        hor_scale = a+8;
        ver_scale = b+b+c+3;
        axis([0 hor_scale 0 ver_scale]);

        %labelling the figure
        eps_x=0.16; % offset in x for dimensioning
        eps_y=0.12; % offset in y for dimensioning

        text(x+a+4*eps_x, y+b/2, 'b','HorizontalAlignment','right'); %labelling b
        text(x+a/2, y-eps_y, 'a','HorizontalAlignment','center'); %labelling a
        text(x+e+d+eps_x+eps_x, y+b+c/2, 'c','HorizontalAlignment','center');
%labelling c
        plot([x+e x+e+d],[y+b+c y+b+c],'k:');%labelling d
        text(x+e+d/2, y+b+c+eps_y+1.5*eps_y, 'd','HorizontalAlignment','center');
%labelling d
        hold off;
```

```
        axis off;


    case 4  %L shaped
        set(handles.text_input,'Value',4);
        a=5;  %width
        b=2;  %height
        c=4;
        x = 4;
        y = 1;
        base_x = [x x+a  x+a  x+b  x+b   x  x ];
        base_y = [y  y   y+b  y+b  y+c  y+c y ];
        plot(base_x, base_y,'-');
        hold on;
        axis([2 a+6 0 c+3]);

        %labelling the figure
        eps_x=0.16; % offset in x
        eps_y=0.12; % offset in y

        text(x+a+eps_x+eps_x+eps_x, y+b/2, 'b','HorizontalAlignment','right');
%labelling b
        text(x+b/2, y+c+eps_y+eps_y, 'b','HorizontalAlignment','right');%labellingb

        text(x+a/2, y-eps_y, 'a','HorizontalAlignment','center'); %labelling a
        text(x-eps_x-eps_x, y+c/2, 'c','HorizontalAlignment','center'); %labellingc
        hold off;
        axis off;


    case 5 %arbitrary cross section
        set(handles.text_input,'Value',5);
        x1=2; x2=3; x3=3.5; x4=3.3; x5=2.4;
        y1=2; y2=3; y3=4.5; y4=6;   y5=8;
        base_x = [x1 x2 x3 x4 x5];
        base_y = [y1 y2 y3 y4 y5];

        plot(base_x, base_y,'-');
        hold on;
        axis([1 4.2 1 11]);

        %labelling the figure
        eps_x=0.16; % offset in x
        eps_y=0.12; % offset in y

        t1=text(x1,y1-eps_y-
eps_y,'(x1,y1)','HorizontalAlignment','center');%labelling x1,y1
        plot(x1,y1,'ko','MarkerSize',2);
        t2=text(x2+3*eps_x,y2-eps_y-
eps_y,'(x2,y2)','HorizontalAlignment','right');%labelling x2,y2
        plot(x2,y2,'ko','MarkerSize',2);

t3=text(x3+4*eps_x,y3+eps_y,'(x3,y3)','HorizontalAlignment','right');%labelling
x3,y3
        plot(x3,y3,'ko','MarkerSize',2);

t4=text(x4+4*eps_x,y4+eps_y,'(x4,y4)','HorizontalAlignment','right');%labelling
x4,y4
        plot(x4,y4,'ko','MarkerSize',2);
        t5=text(x5,y5+3*eps_y,'(x5,y5)','HorizontalAlignment','center');%labelling
x5,y5
        plot(x5,y5,'ko','MarkerSize',2);
        hold off;
        axis off;

end
```

```
set(handles.pushbutton_input,'Visible','On');  % enabling the pushbutton 'press to
provide inputs'

guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function pushbutton_preview_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pushbutton_preview (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function axes_displaycrosssection_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes_displaycrosssection (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
%imshow(preview.jpg);
% Hint: place code in OpeningFcn to populate axes_displaycrosssection


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
pop_crosssectiontype.
function pop_crosssectiontype_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to pop_crosssectiontype (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object deletion, before destroying properties.
function axes_displaycrosssection_DeleteFcn(hObject, eventdata, handles)
% hObject    handle to axes_displaycrosssection (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function text_preview_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text_preview (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on button press in pushbutton_input.
function pushbutton_input_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%setting callback for button named "press to provide inputs". After
%clicking this button,following function runs setting up(enabling) the panel for
receiving user inputs
val=get(handles.text_input,'Value');
switch val
    case 1 %rectangle
        set(handles.text_I_a,'Enable','on');
        set(handles.text_I_b,'Enable','on');
        set(handles.text_I_c,'Enable','off');
        set(handles.text_I_d,'Enable','off');
        set(handles.edit_I_a,'Enable','on');
        set(handles.edit_I_b,'Enable','on');
        set(handles.edit_I_c,'Enable','off');
        set(handles.edit_I_d,'Enable','off');
        set(handles.pushbutton_I_submit,'Enable','on');
        set(handles.edit_arbit_x,'Enable','off');
```

```
    set(handles.edit_arbit_y,'Enable','off');
    set(handles.text_arbit_x,'Enable','off');
    set(handles.text_arbit_y,'Enable','off');
    set(handles.text_arbit_top,'Enable','off');
    set(handles.pushbutton_arbit_submit,'Enable','off');
    set(handles.pushbutton_arbit_next,'Enable','off');
case 2 %triangle
     set(handles.text_I_a,'Enable','on');
    set(handles.text_I_b,'Enable','on');
    set(handles.text_I_c,'Enable','off');
    set(handles.text_I_d,'Enable','off');
    set(handles.edit_I_a,'Enable','on');
    set(handles.edit_I_b,'Enable','on');
    set(handles.edit_I_c,'Enable','off');
    set(handles.edit_I_d,'Enable','off');
    set(handles.pushbutton_I_submit,'Enable','on');
    set(handles.edit_arbit_x,'Enable','off');
    set(handles.edit_arbit_y,'Enable','off');
    set(handles.text_arbit_x,'Enable','off');
    set(handles.text_arbit_y,'Enable','off');
    set(handles.text_arbit_top,'Enable','off');
    set(handles.pushbutton_arbit_submit,'Enable','off');
    set(handles.pushbutton_arbit_next,'Enable','off');
case 3 %I shaped
     set(handles.text_I_a,'Enable','on');
    set(handles.text_I_b,'Enable','on');
    set(handles.text_I_c,'Enable','on');
    set(handles.text_I_d,'Enable','on');
    set(handles.edit_I_a,'Enable','on');
    set(handles.edit_I_b,'Enable','on');
    set(handles.edit_I_c,'Enable','on');
    set(handles.edit_I_d,'Enable','on');
    set(handles.pushbutton_I_submit,'Enable','on');
    set(handles.edit_arbit_x,'Enable','off');
    set(handles.edit_arbit_y,'Enable','off');
    set(handles.text_arbit_x,'Enable','off');
    set(handles.text_arbit_y,'Enable','off');
    set(handles.text_arbit_top,'Enable','off');
    set(handles.pushbutton_arbit_submit,'Enable','off');
    set(handles.pushbutton_arbit_next,'Enable','off');
case 4 %L shaped
    set(handles.text_I_a,'Enable','on');
    set(handles.text_I_b,'Enable','on');
    set(handles.text_I_c,'Enable','on');
    set(handles.text_I_d,'Enable','off');
    set(handles.edit_I_a,'Enable','on');
    set(handles.edit_I_b,'Enable','on');
    set(handles.edit_I_c,'Enable','on');
    set(handles.edit_I_d,'Enable','off');
    set(handles.pushbutton_I_submit,'Enable','on');
    set(handles.edit_arbit_x,'Enable','off');
    set(handles.edit_arbit_y,'Enable','off');
    set(handles.text_arbit_x,'Enable','off');
    set(handles.text_arbit_y,'Enable','off');
    set(handles.text_arbit_top,'Enable','off');
    set(handles.pushbutton_arbit_submit,'Enable','off');
    set(handles.pushbutton_arbit_next,'Enable','off');
case 5 %arbitrary shaped
    set(handles.text_I_a,'Enable','off');
    set(handles.text_I_b,'Enable','off');
    set(handles.text_I_c,'Enable','off');
    set(handles.text_I_d,'Enable','off');
    set(handles.edit_I_a,'Enable','off');
    set(handles.edit_I_b,'Enable','off');
    set(handles.edit_I_c,'Enable','off');
    set(handles.edit_I_d,'Enable','off');
```

```
        set(handles.pushbutton_I_submit,'Enable','off');
        set(handles.edit_arbit_x,'Enable','on');
        set(handles.edit_arbit_y,'Enable','on');
        set(handles.text_arbit_x,'Enable','on');
        set(handles.text_arbit_y,'Enable','on');
        set(handles.text_arbit_top,'Enable','on');
        set(handles.pushbutton_arbit_submit,'Enable','off');
        set(handles.pushbutton_arbit_next,'Enable','on');
    otherwise
end




function edit_I_a_Callback(hObject, eventdata, handles)
% hObject    handle to edit_I_a (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_I_a as text
%        str2double(get(hObject,'String')) returns contents of edit_I_a as a double


% --- Executes during object creation, after setting all properties.
function edit_I_a_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_I_a (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_I_b_Callback(hObject, eventdata, handles)
% hObject    handle to edit_I_b (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_I_b as text
%        str2double(get(hObject,'String')) returns contents of edit_I_b as a double


% --- Executes during object creation, after setting all properties.
function edit_I_b_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_I_b (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_I_c_Callback(hObject, eventdata, handles)
% hObject    handle to edit_I_c (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_I_c as text
%        str2double(get(hObject,'String')) returns contents of edit_I_c as a double


% --- Executes during object creation, after setting all properties.
function edit_I_c_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_I_c (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_I_d_Callback(hObject, eventdata, handles)
% hObject    handle to edit_I_d (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_I_d as text
%        str2double(get(hObject,'String')) returns contents of edit_I_d as a double


% --- Executes during object creation, after setting all properties.
function edit_I_d_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_I_d (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_arbit_x_Callback(hObject, eventdata, handles)
% hObject    handle to edit_arbit_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_arbit_x as text
%        str2double(get(hObject,'String')) returns contents of edit_arbit_x as a
double


% --- Executes during object creation, after setting all properties.
function edit_arbit_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_arbit_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


function edit_arbit_y_Callback(hObject, eventdata, handles)
% hObject    handle to edit_arbit_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_arbit_y as text
%        str2double(get(hObject,'String')) returns contents of edit_arbit_y as a
double


% --- Executes during object creation, after setting all properties.
function edit_arbit_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_arbit_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




% --- Executes on button press in pushbutton_arbit_next.
function pushbutton_arbit_next_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_arbit_next (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for the 'next coordinate' button provided within the panel
taking
%inputs for arbitrary cross-sections. Clicking the button runs following
%set of codes. It accepts user input for x and y coordinates (for arbitrary cross
sections) and enables
%the submit button upon entry of 3 coordinate sets
x=str2num(get(handles.edit_arbit_x,'String'));
y=str2num(get(handles.edit_arbit_y,'String'));
set(handles.edit_arbit_x,'String','');
set(handles.edit_arbit_y,'String','');
if ~isempty(x)&&~isempty(y)
    a=get(handles.text_x_array,'Value');
    b=get(handles.text_y_array,'Value');
    set(handles.text_x_array,'Value',[a x]);
    set(handles.text_y_array,'Value',[b y]);
    a=get(handles.text_x_array,'Value');
    b=get(handles.text_y_array,'Value');
    if size(a,2)==3&&size(b,2)==3
        set(handles.pushbutton_arbit_submit,'Enable','On');
    end

end


% --- Executes on button press in pushbutton_nextpage.
function pushbutton_nextpage_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_nextpage (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

%callback function for the button 'confirm and go to next page'.following
%function executes after pressing above button.Stores the value of key
%parameters such as a,b,c,d,x_array,y_array in the root directory to be
%used in 2nd page
%It also closes the current page and opens up  page 2
x1=get(handles.text_x_array,'Value');
y1=get(handles.text_y_array,'Value');
set(handles.text_x_array,'Value',x1);
set(handles.text_y_array,'Value',y1);
x=get(handles.text_x_array,'Value');
y=get(handles.text_y_array,'Value');
a=get(handles.text_a,'Value');
b=get(handles.text_b,'Value');
c=get(handles.text_c,'Value');
d=get(handles.text_d,'Value');
crosssection=get(handles.text_input,'Value');
setappdata(0,'a',a); setappdata(0,'b',b); setappdata(0,'c',c);
setappdata(0,'d',d); setappdata(0,'x',x); setappdata(0,'y',y);
setappdata(0,'crosssection',crosssection);
close(input_gui);
page_2_gui;




% --- Executes on button press in pushbutton_I_submit.
function pushbutton_I_submit_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_I_submit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for the button 'submit' in the panel accepting values for
a,b,c,d.following
%function executes after pressing above button. Accepts user input in form
%of a,b,c,d. It disable various edit, text fields and submit buttons and
%enables Reset and nextpage buttons
 a=str2num(get(handles.edit_I_a,'String'));
 b=str2num(get(handles.edit_I_b,'String'));
 c=str2num(get(handles.edit_I_c,'String'));
 d=str2num(get(handles.edit_I_d,'String'));
 set(handles.text_a,'Value',a);
 set(handles.text_b,'Value',b);
 set(handles.text_c,'Value',c);
 set(handles.text_d,'Value',d);

 handle_array = [handles.pop_crosssectiontype, handles.pushbutton_preview,
handles.pushbutton_input, ...
     handles.text_I_a, handles.text_I_b, handles.text_I_c, handles.text_I_d,
handles.text_arbit_x...
    ,handles.text_arbit_y, handles.text_arbit_top, handles.edit_I_a,
handles.edit_I_b, handles.edit_I_c, handles.edit_I_d...
    ,handles.edit_arbit_x, handles.edit_arbit_y, handles.pushbutton_I_submit,
handles.pushbutton_arbit_next, handles.pushbutton_arbit_submit];
 set(handle_array,'Enable','Off');    %disabling text fields
 enable_array = [handles.pushbutton_reset,handles.pushbutton_nextpage];
 set(enable_array,'Enable','on');     %enabling reset and next page buttons

% --- Executes on button press in pushbutton_arbit_submit.
function pushbutton_arbit_submit_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_arbit_submit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
%callback function for the button 'submit' in the panel accepting values for
arbitrary cross-section .following
%function executes after pressing above button. disables all text,edit
%fields and enables  reset and next page buttons
x=str2num(get(handles.edit_arbit_x,'String'));
y=str2num(get(handles.edit_arbit_y,'String'));
if ~isempty(x)&&~isempty(y)
    a=get(handles.text_x_array,'Value');
    b=get(handles.text_y_array,'Value');
    set(handles.text_x_array,'Value',[a x]);
    set(handles.text_y_array,'Value',[b y]);
end

handle_array = [handles.pop_crosssectiontype, handles.pushbutton_preview,
handles.pushbutton_input, ...
     handles.text_I_a, handles.text_I_b, handles.text_I_c, handles.text_I_d,
handles.text_arbit_x...
    ,handles.text_arbit_y, handles.text_arbit_top, handles.edit_I_a,
handles.edit_I_b, handles.edit_I_c, handles.edit_I_d...
    ,handles.edit_arbit_x, handles.edit_arbit_y, handles.pushbutton_I_submit,
handles.pushbutton_arbit_next, handles.pushbutton_arbit_submit];
 set(handle_array,'Enable','Off');    %disabling text fields
 enable_array = [handles.pushbutton_reset,handles.pushbutton_nextpage];
 set(enable_array,'Enable','on');     %enabling reset and next page buttons


% --- Executes on button press in pushbutton_reset.
function pushbutton_reset_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_reset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for the button 'Reset'.following
%function executes after pressing above button. Reinitializes the first
%page
 enable_array = [handles.pop_crosssectiontype, handles.pushbutton_preview,
handles.pushbutton_input];
clear_edit_array = [handles.edit_I_a, handles.edit_I_b, handles.edit_I_c,
handles.edit_I_d, handles.edit_arbit_x, handles.edit_arbit_y];
clear_text_array = [handles.text_x_array, handles.text_y_array,
handles.text_a,handles.text_b,handles.text_c,handles.text_d];
 set(enable_array,'Enable','On');
 set(clear_edit_array,'String','');
 set(clear_text_array,'Value',[]);
 set(handles.pushbutton_nextpage,'Enable','off');   %disabling text fields and
pushbuttons to reinitialize input page
 set(handles.pushbutton_reset,'Enable','off');


% --- Executes during object creation, after setting all properties.
function pushbutton_nextpage_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pushbutton_nextpage (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function edit20_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

Code for page_2_gui Function (Page 2 of GUI)

```
function varargout = page_2_gui(varargin)
% PAGE_2_GUI MATLAB code for page_2_gui.fig
%      PAGE_2_GUI, by itself, creates a new PAGE_2_GUI or raises the existing
%      singleton*.
%
%    H = PAGE_2_GUI returns the handle to a new PAGE_2_GUI or the handle to
%      the existing singleton*.
%
% PAGE_2_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in PAGE_2_GUI.M with the given input arguments.
%
% PAGE_2_GUI('Property','Value',...) creates a new PAGE_2_GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before page_2_gui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to page_2_gui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help page_2_gui

% Last Modified by GUIDE v2.5 11-Nov-2017 01:25:55

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @page_2_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @page_2_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before page_2_gui is made visible.
function page_2_gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to page_2_gui (see VARARGIN)

% Choose default command line output for page_2_gui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
% UIWAIT makes page_2_gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);

%Generating the intial view for the window by disabling unnecessary edit,text
fields and pushbuttons
handle1_array=[handles.text_input_Mx, handles.text_input_My, handles.edit_input_Mx,
handles.edit_input_My, handles.pushbutton_submit...
    ,handles.pushbutton_next];
set(handle1_array,'Enable','off');

% --- Outputs from this function are returned to the command line.
function varargout = page_2_gui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton_plot.
function pushbutton_plot_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_plot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for the button 'Click to input moment values'. Following
%code executes after pressing above button. It initializes the page by taking up
the parameters 'a,b,c,d,x,y' stored in the root
%file by page 1(input page).Through the parameters and the case of selected
%cross section, it sets in the vectors plot_x and plot_y to be used for showing the
image of cross-section and principle axes
 handle1_array=[handles.text_input_Mx, handles.text_input_My,
handles.edit_input_Mx, handles.edit_input_My,handles.pushbutton_submit];
 set(handle1_array,'Enable','on');
 a=getappdata(0,'a'); b=getappdata(0,'b'); c=getappdata(0,'c');
d=getappdata(0,'d');e=(a-d)/2;...
 x_array=getappdata(0,'x'); y_array=getappdata(0,'y');
 crosssection=getappdata(0,'crosssection'); % case of selected cross-section
 plot_x=[];   %vectors containing the vertices of convex polygon enetred as input
 plot_y=[];   % also used for plotting
 x1=0;
 y1=0;
 switch crosssection
     case 1  %rectangular case
        plot_x = [x1 x1+a x1+a  x1  x1];
```

```
        plot_y = [y1  y1  y1+b y1+b y1];
     case 2 %triangular case
        plot_x = [x1 x1+a x1+a/2 x1 ];
        plot_y = [y1 y1 y1+b y1];
     case 3  %I shaped
        plot_x = [x1 x1+a  x1+a  x1+a-e  x1+a-e   x1+a    x1+a     x1       x1
x1+e  x1+e x1   x1 ];  % x coordinates (anticlockwise)
        plot_y = [y1  y1   y1+b  y1+b    y1+b+c  y1+b+c y1+b+c+b y1+b+c+b y1+b+c
y1+b+c y1+b y1+b y1 ];  % x coordinates (anticlockwise)
     case 4  %L shaped
        plot_x = [x1 x1+a  x1+a  x1+b  x1+b   x1  x1 ];
        plot_y = [y1  y1   y1+b  y1+b  y1+c  y1+c y1 ];
     case 5  %arbitrary cross-section
        plot_x = [x_array x_array(1)];
        plot_y = [y_array y_array(1)];
     otherwise
 end


 % retrieving geometrical values and plotting

 [geom,iner,cpmo,eig_vec, xm, ym ]=polygeom(plot_x,plot_y); %using the function
polygeom(included in the base code),geometrical
 X_C=geom(2);    % x coordinate of centroid                  %parameters like
centroid, moments of inertias are retrieved
 Y_C=geom(3);    % y coordinate of centroid
 ang1=cpmo(2); ang2=cpmo(4);

 x_mag=max(plot_x)-min(plot_x);
 y_mag=max(plot_y)-min(plot_y);
 principle_axes_mag=max(x_mag,y_mag);

 % creating vectors (Mx_vec and My_vec) to be used for plotting the priciple axes
 x_principle_1 = [ X_C-principle_axes_mag*cos(ang1)
X_C+principle_axes_mag*cos(ang1) ];
 y_principle_1 = [ Y_C-principle_axes_mag*sin(ang1)
Y_C+principle_axes_mag*sin(ang1) ];

 x_principle_2 = [ X_C-principle_axes_mag*cos(ang2)
X_C+principle_axes_mag*cos(ang2) ];
 y_principle_2 = [ Y_C-principle_axes_mag*sin(ang2)
Y_C+principle_axes_mag*sin(ang2) ];

 Mx_vec = [principle_axes_mag*cos(ang1)/2 principle_axes_mag*sin(ang1)/2];
 My_vec = [principle_axes_mag*cos(ang2)/2 principle_axes_mag*sin(ang2)/2];

 %storing vectors used for plotting principle axes in the root directory(to be used
in page_3)
 setappdata(0,'Mx_vec',Mx_vec);
 setappdata(0,'My_vec',My_vec);
 setappdata(0,'X_C',X_C);
 setappdata(0,'Y_C',Y_C);
 setappdata(0,'plot_x',plot_x);
 setappdata(0,'plot_y',plot_y);
 setappdata(0,'x_principle_1',x_principle_1);
 setappdata(0,'y_principle_1',y_principle_1);
 setappdata(0,'x_principle_2',x_principle_2);
 setappdata(0,'y_principle_2',y_principle_2);

 %plotting figure of selected cross-section along with the principle axes
 plot(plot_x, plot_y,'b-');
 hold on;
 plot(X_C,Y_C,'ro', x_principle_1,y_principle_1,'k:',
x_principle_2,y_principle_2,'k:');
 quiver(X_C,Y_C,Mx_vec(1),Mx_vec(2),'r');
 quiver(X_C,Y_C,My_vec(1),My_vec(2),'r');
```

```
 xlabel('x axis');
 ylabel('y axis');
 axis square;
```

```
function edit_input_Mx_Callback(hObject, eventdata, handles)
% hObject    handle to edit_input_Mx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_input_Mx as text
%        str2double(get(hObject,'String')) returns contents of edit_input_Mx as a
double


% --- Executes during object creation, after setting all properties.
function edit_input_Mx_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_input_Mx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_input_My_Callback(hObject, eventdata, handles)
% hObject    handle to edit_input_My (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_input_My as text
%        str2double(get(hObject,'String')) returns contents of edit_input_My as a
double

% --- Executes during object creation, after setting all properties.
function edit_input_My_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_input_My (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton_previous.
function pushbutton_previous_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_previous (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for the button named 'Previous Page'.Following code runs upon
clicking above button.
%This function closes current page(page_2) and opens up page1(input_gui)
```

```
close(page_2_gui);
input_gui;


% --- Executes on button press in pushbutton_next.
function pushbutton_next_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_next (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for the button named 'Confirm and go to next page'.Following
code runs upon clicking above button.
%saves the value of Mx and My in root directory(to be used in page_3).
%closes current page and opens page_3
Mx=str2num(get(handles.edit_input_Mx,'String'));
My=str2num(get(handles.edit_input_My,'String'));
setappdata(0,'Mx_page3',Mx);
setappdata(0,'My_page3',My);
close(page_2_gui)
page_3_gui;


% --- Executes on button press in pushbutton_submit.
function pushbutton_submit_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_submit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for the button named 'submit'.Following code runs upon clicking
above button.
%Accepts the user input Mx and My. Disables unnecessary text and edit
%fields.
Mx=str2num(get(handles.edit_input_Mx,'String'));
My=str2num(get(handles.edit_input_My,'String'));
if ~isempty(Mx)&&~isempty(My)
  setappdata(0,'Mx_page3',Mx);
  setappdata(0,'Mx_page3',My);
  handle1_array=[handles.text_input_Mx, handles.text_input_My,
handles.edit_input_Mx, handles.edit_input_My, handles.pushbutton_submit];
  set(handle1_array,'Enable','off');
  set(handles.pushbutton_next,'Enable','on');
end
```

Code for page_3_gui Function (Page 3 in GUI)

```
function varargout = page_3_gui(varargin)
% PAGE_3_GUI MATLAB code for page_3_gui.fig
%      PAGE_3_GUI, by itself, creates a new PAGE_3_GUI or raises the existing
%      singleton*.
%
%      H = PAGE_3_GUI returns the handle to a new PAGE_3_GUI or the handle to
%      the existing singleton*.
%
%      PAGE_3_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PAGE_3_GUI.M with the given input arguments.
%
%      PAGE_3_GUI('Property','Value',...) creates a new PAGE_3_GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before page_3_gui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to page_3_gui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help page_3_gui

% Last Modified by GUIDE v2.5 12-Nov-2017 18:37:57

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @page_3_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @page_3_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before page_3_gui is made visible.
function page_3_gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to page_3_gui (see VARARGIN)

% Choose default command line output for page_3_gui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
```

```
%Generating the intial view for the window by disabling unnecessary edit,text
fields and pushbuttons
handle2_array = [handles.text_x, handles.text_y, handles.edit_x, handles.edit_y,
handles.pushbutton_submit_reenter, handles.edit_value...
    , handles.text_value, handles.pushbutton_complete, handles.pushbutton_point,
handles.pushbutton_reenter, handles.text_Mx,...
    handles.text_My, handles.edit_Mx, handles.edit_My,
handles.pushbutton_calculate];
set(handle2_array,'Enable','off');
% retreiving key Mx and My stored in root directory and storing as M vector
% to be used in page 3
 Mx=getappdata(0,'Mx_page3');
 My=getappdata(0,'My_page3');
 M=[Mx My];
 setappdata(0,'M',M);


% UIWAIT makes page_3_gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = page_3_gui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton_complete.
function pushbutton_complete_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_complete (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for the button 'Complete Stress Distribution'. Following
%code executes after pressing above button. Generates a 3D plot using
%the function 'quiver3' containing the vectors depicting
%nature(tensile/compressive) and magnitude of stress at interior points of polygon
plot_x=getappdata(0,'plot_x');                                 %receiveing values
of plot_x,plot_y and M from the root directory
plot_y=getappdata(0,'plot_y');
M=getappdata(0,'M');
[Stress,grid_x,grid_y] = Stress_calculator(plot_x,plot_y,M);    %calculating stress
at decided number of points within the figure
p=size(grid_x,1);                                              %the function being
used is Stress_calculator.Input for this function
q=size(grid_x,2);                                              %are vectors
plot_x,plot_y containing vertices of polygon in anticlockwise sense
% figure;                                                      %and vector M
containing components of moment vector in x and y direction


%plotting vectors for stress values at each decided point
quiver3(grid_x,grid_y,zeros(p,q),zeros(p,q),zeros(p,q),Stress );  %intiates a 3D
plot                                                                %section
hold on;
rotate3d on;

plot(plot_x,plot_y,'k-');                   %plotting the cross-section in the same
figure
xlabel('x axis');
ylabel('y axis');
```

```
Mx_vec=getappdata(0,'Mx_vec');
My_vec=getappdata(0,'My_vec');
X_C=getappdata(0,'X_C');
Y_C=getappdata(0,'Y_C');
quiver(X_C,Y_C,Mx_vec(1),Mx_vec(2),'r');   %plotting priciple axes in the same
figure
quiver(X_C,Y_C,My_vec(1),My_vec(2),'r');
hold off;

% % Neutral axis
% i=Stress(Stress==0);

% --- Executes on button press in pushbutton_point.
function pushbutton_point_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_point (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for the button 'Stress at a point'. Following
%code executes after pressing above button. Enables corresponding text and
%edit fields
handle2_array = [handles.text_x, handles.text_y, handles.edit_x, handles.edit_y,
handles.pushbutton_calculate, handles.edit_value...
    , handles.text_value];
set(handle2_array,'Enable','on');



function edit_x_Callback(hObject, eventdata, handles)
% hObject    handle to edit_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_x as text
%        str2double(get(hObject,'String')) returns contents of edit_x as a double


% --- Executes during object creation, after setting all properties.
function edit_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_y_Callback(hObject, eventdata, handles)
% hObject    handle to edit_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_y as text
%        str2double(get(hObject,'String')) returns contents of edit_y as a double


% --- Executes during object creation, after setting all properties.
function edit_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_value_Callback(hObject, eventdata, handles)
% hObject    handle to edit_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_value as text
%        str2double(get(hObject,'String')) returns contents of edit_value as a
double


% --- Executes during object creation, after setting all properties.
function edit_value_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton_submit_reenter.
function pushbutton_submit_reenter_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_submit_reenter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for the button 'submit'. Following
%code executes after pressing above button. It accepts the input for new Mx and
%My values and updates vector M to be used for calculating Stress at each
%interior point and visualizing
%stress distribution
Mx=str2num(get(handles.edit_Mx,'String'));
My=str2num(get(handles.edit_My,'String'));
M=[Mx My];
setappdata(0,'M',M);
handle_disable_array=[handles.text_Mx, handles.text_My, handles.edit_Mx,
handles.edit_My, handles.pushbutton_submit_reenter];
handle_enable_array=[handles.pushbutton_point, handles.pushbutton_complete];
handle_clear_array=[handles.edit_Mx, handles.edit_My];
set(handle_clear_array,'String','');
set(handle_enable_array,'Enable','on');
set(handle_disable_array,'Enable','off');

% S=Stress_point(plot_x,plot_y,[x,y],M);       %The function Stress_point has been
made by the coder. It takes input in form of vectors
% set(handles.edit_value,'String',num2str(S)); %containing vertices of ploygon
along with the point at which stress has to be calculated
                                             %and the moment vector


% --- Executes on button press in pushbutton_click.
```

```
function pushbutton_click_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_click (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for the button 'Click to proceed'. Following
%code executes after pressing above button. Enables pushbuttons
%'Stress at a point' and 'Complete Stress distribution'. Retreives
%parameters 'plot_x,plot_y,X_C,Y_C,Mx_vec,My_vec' etc saved in  the root
%directory and draws the figure showing selected cross-section and
%principle axes
handle_array=[handles.pushbutton_complete, handles.pushbutton_point,
handles.pushbutton_reenter];
set(handle_array,'Enable','on');
set(handles.pushbutton_click,'Enable','off');

Mx_vec=getappdata(0,'Mx_vec');
My_vec=getappdata(0,'My_vec');
X_C=getappdata(0,'X_C');
Y_C=getappdata(0,'Y_C');
plot_x=getappdata(0,'plot_x');
plot_y=getappdata(0,'plot_y');
x_principle_1=getappdata(0,'x_principle_1');
y_principle_1=getappdata(0,'y_principle_1');
x_principle_2=getappdata(0,'x_principle_2');
y_principle_2=getappdata(0,'y_principle_2');

plot(plot_x, plot_y,'b-');
hold on;
plot(X_C,Y_C,'ro', x_principle_1,y_principle_1,'k:',
x_principle_2,y_principle_2,'k:');
quiver(X_C,Y_C,Mx_vec(1),Mx_vec(2),'r');
quiver(X_C,Y_C,My_vec(1),My_vec(2),'r');
xlabel('x axis');
ylabel('y axis');
axis square;
hold off;


% --- Executes on button press in pushbutton_page1.
function pushbutton_page1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_page1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for the button 'Go to page1'. Following
%code executes after pressing above button.
%closes the current page and opens up page_1(input_gui)
close(page_3_gui);
input_gui;


% --- Executes on button press in pushbutton_reenter.
function pushbutton_reenter_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_reenter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for the button 'Reenter moment values'. Following
%code executes after pressing above button.Enables the edit fields used to
%take inputs for new values of Mx and My along with showing the cross
%section
handle_disable_array=[handles.pushbutton_point, handles.text_x, handles.text_y,
handles.edit_x, handles.edit_y, handles.pushbutton_calculate...
    handles.text_value, handles.edit_value, handles.pushbutton_complete];
```

```
handle_enable_array=[handles.text_Mx, handles.text_My, handles.edit_Mx,
handles.edit_My, handles.pushbutton_submit_reenter];
handle_clear_array=[handles.edit_x, handles.edit_y,handles.edit_value];
set(handle_disable_array,'Enable','off');
set(handle_enable_array,'Enable','on');
set(handle_clear_array,'String','');

Mx_vec=getappdata(0,'Mx_vec');
My_vec=getappdata(0,'My_vec');
X_C=getappdata(0,'X_C');
Y_C=getappdata(0,'Y_C');
plot_x=getappdata(0,'plot_x');
plot_y=getappdata(0,'plot_y');
x_principle_1=getappdata(0,'x_principle_1');
y_principle_1=getappdata(0,'y_principle_1');
x_principle_2=getappdata(0,'x_principle_2');
y_principle_2=getappdata(0,'y_principle_2');

plot(plot_x, plot_y,'b-');
hold on;
plot(X_C,Y_C,'ro', x_principle_1,y_principle_1,'k:',
x_principle_2,y_principle_2,'k:');
quiver(X_C,Y_C,Mx_vec(1),Mx_vec(2),'r');
quiver(X_C,Y_C,My_vec(1),My_vec(2),'r');
xlabel('x axis');
ylabel('y axis');
axis square;
hold off;



function edit_Mx_Callback(hObject, eventdata, handles)
% hObject    handle to edit_Mx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_Mx as text
%        str2double(get(hObject,'String')) returns contents of edit_Mx as a double


% --- Executes during object creation, after setting all properties.
function edit_Mx_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_Mx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_My_Callback(hObject, eventdata, handles)
% hObject    handle to edit_My (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_My as text
%        str2double(get(hObject,'String')) returns contents of edit_My as a double


% --- Executes during object creation, after setting all properties.
```

```
function edit_My_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_My (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function pushbutton_reenter_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pushbutton_reenter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on button press in pushbutton_calculate.
function pushbutton_calculate_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_calculate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%callback function for the button 'calculate'. Following
%code executes after pressing above button. Accepts the input given for
%calculating stress at a point. Retrives parameters such as 'plot_x, plot_y'
%from the root directory and calculates value of stress at the point asked
plot_x=getappdata(0,'plot_x');
plot_y=getappdata(0,'plot_y');
M=getappdata(0,'M');
x=str2num(get(handles.edit_x,'String'));
y=str2num(get(handles.edit_y,'String'));
S=num2str(Stress_point(plot_x,plot_y,[x y],M));
set(handles.edit_value,'String',S);
```

Code for Stress_point Function

```
function S = Stress_point(x,y,point,M)
% this code is primarily the base code named 'test.m' made by the coder . As
% a visualizer, I have just added few lines of code in it.


% Polygonal Cross Section; Symmetric as well as Unsymmetric Pure Bending
% Scenario with Moment vector given at the cross section with phi as the
% angle between the plane of load and x-z plane; phi is taken positive
% counter-clockwise

% constants

d2r = pi / 180;
x=x(1:end-1);
y=y(1:end-1);

[geom,iner,cpmo,eig_vec, xm, ym ]=polygeom(x,y);
%applying the function defined by the name polygeom in the polygeom.m file.
%Here x and y are the vectors representing the x and y coordinates of the
%points on the arbitrary polygonal cross section. This points are necessary
%to feed the algorithm, information about the geometry of the cross
%section.
theta=atan(M(2)/M(1));
xx=point(1,1);
yy=point(1,2);
A=geom(1);
X_C=geom(2);
Y_C=geom(3);
P=geom(4);
%geom = [ area   X_cen  Y_cen  perimeter ]
%iner = [ Ixx    Iyy    Ixy    Iuu    Ivv    Iuv ]
%u,v are centroidal axes parallel to x,y axes.
%cpmo= [ I1     ang1   I2     ang2   J ]
%I1,I2 are centroidal principal moments about axes
%at angles ang1,ang2.
%ang1 and ang2 are in radians.
%J is centroidal polar moment.
%J = I1 + I2 = Iuu + Ivv

I_xx=iner(1);
I_yy=iner(2);
I_xy=iner(3);
I_uu=iner(4);
I_vv=iner(5);
I_uv=iner(6);

I1=cpmo(1);
ang1=cpmo(2);
I2=cpmo(3);
ang2=cpmo(4);
J=cpmo(5);



% Once we know the Principal Moments of Inertia what we can do is to
% project the Moment Vector acting on a given cross section along the
% direction of the principal axis.
% Thereafter calculating the norms of the respective projected vectors,
% we get M_1 and M_2.
% Now we can just plug in the above calculated quantities in the formula to
% determine stress at a particular point on the cross section.
% The necessary formula is mentioned as Equation (7.4) in Chapter 7 Bending
```

```
% of Straight Beams, Page No. 268 in Advanced Mechanics of Materials,
% 6th Edition Arthur P. Boresi, Richard J. Schmidt; ISBN: 978-0-471-43881-6
% if ang1*theta<0
%     gamma1=ang1+theta;
% end
% if ang1*theta>0
%     gamma1=ang1-theta;
% end
%
% if ang2*theta<0
%     gamma2=ang2+theta;
% end
% if ang2*theta>0
%     gamma2=ang2-theta;
% end
%
% if theta ==0
%     gamma1=ang1;
%     gamma2=ang2;
% end

M_11=proj(M,[eig_vec(1,1), eig_vec(2,1)]);
M_22=proj(M,[eig_vec(1,2), eig_vec(2,2)]);
M_1=norm(M_11);
M_2=norm(M_22);
M_111=norm(M)*cos(theta+ang1);
M_222=norm(M)*cos((d2r*90)+ang1+theta);

% shift the coordinates from the specified x-y coordinate frame defined by
% the user when she/he was giving the coordinates of the cross section as
% input to the centroidal coordinate frame and then to the Principal Axis
% Coordinate Frame by transforming the coordinates by angle ang1.
% We are doing this as X and Y coordinates of the point at which the user
% requires the Stress is wrt the coordinate axis defined by the user
% whereas in the Equation (7.4) in Chapter 7 Bending
% of Straight Beams, Page No. 268 in Advanced Mechanics of Materials,
% 6th Edition Arthur P. Boresi, Richard J. Schmidt; ISBN: 978-0-471-43881-6
% x and y coordinates (in the formula) is wrt thw Principal Axis.
xx=xx-X_C;
yy=yy-Y_C;
XT=(xx*cos(ang1))+(yy*sin(ang1));
YT=(-1*xx*sin(ang1))+(yy*cos(ang1));
if ang1==0
    S=((((M_1*yy)/I1)-((M_2*xx)/I2))*1000);
elseif ang2==0
    S=-1*((((M_1*xx)/I1)-((M_2*yy)/I2))*1000);
else
    S=((((M_111*YT)/I1)-((M_222*XT)/I2))*10000);
%     phi=input('Enter phi (in degrees): ');
%     phi=d2r*phi;
%     Mx=norm(M)*sin(phi);
%     ta=(I_uv-(I_uu*cot(phi)))/(I_vv-(I_uv*cot(phi)));
%     S=(Mx*(yy-(xx*ta)))/I_uu-(I_uv*ta);
end
```

Code for Stress_calculator Function

```
function [Stress,rect_x,rect_y] = Stress_calculator(x,y,M)
% Inputs are the vertices of convex polygon and moment vector .function returns
three matrices.
% rect_x rect_y are the meshgrid representing 'check rectangle'- the rectangle
surrounding the
% input convex polygon. This rectangle is scanned by the function named
% 'inpolygon' to get all the points which lie withinn the convex polygon
% whose vertices are provided as inputs.
% Then using the input vector M and function stress_point , stresses at
% dicretized interior points are calculated and returned in form of matrix
% Stress

% creating rectangular area to check for (rect_x and rect_y)

% eps=5;   % discretization
n=10;  %number of vectors to be used for plotting in x and y directions
eps_x=(max(x)-min(x))/(n)
eps_y=(max(y)-min(y))/(n)
points = [min(x)-eps_x  max(x)+eps_x  min(y)-eps_y  max(y)+eps_y];
rect_x=[];
rect_y=[];
for i=points(3):eps_y:points(4)
    rect_x=[rect_x;points(1):eps_x:points(2)];
end
j=zeros(1,size(points(1):eps_x:points(2),2));
for i=points(3):eps_y:points(4)
    rect_y=[rect_y;j+i];
end

% checking interior points for convex polygon
in=inpolygon(rect_x,rect_y,x,y);   %inpolygon takes input form of four vectors. 2
are the matrices representing
                                   %coordinates to scan for. The second two are the
vectors representing vertices
                                   %of test polygon
                                   %returned value is a logical array 'in'
                                   %which represents the points to consider inside
rect_x and rect_y

%creating grid for interior points
rect=cellfun(@(x,y) [x y],num2cell(rect_x)',num2cell(rect_y)','un',0);
% creates a cell array 'rect' containing coordinates of
rect=rect';
 % interior points of the polygon reference for the function used

 % reference

https://in.mathworks.com/matlabcentral/answers



% calculating value of stress at interior points
Stress = zeros(size(rect_x));
a=cell2mat(rect(in==1));         %converts the cell array 'rect' to matrix form. 'a'
contains the coordinates
b=zeros(size(a,1),1);            %of interior points in matrix form
for i=1:size(a,1)
    z=[a(i,1) a(i,2)];
    b(i)=Stress_point( x,y,z,M);
end
```

```
Stress(in==1)= b;      % Stress is a matrix equivalent to the size of matrix rect_x
or rect_y. Value of stress
                       % at the exterior points are 0.
```