

Mini Project

Team: Meet Gandhi 15110049 Darshan Patel 15110083 Rajat Ranjan 15110098 Saksham Singal 15110112

Stresses in Beams



[1]

Table of Contents

[Introduction: 2](#_Toc498292473)

[Why this project? 2](#_Toc498292474)

[Theory: 2](#_Toc498292475)

[Introduction 2](#_Toc498292476)

[Some basic concepts 2](#_Toc498292477)

[Pure Bending Scenario 4](#_Toc498292478)

[Deriving the flexural formula 4](#_Toc498292479)

[Using the flexural formula 6](#_Toc498292480)

[Working of the Software 6](#_Toc498292481)

[About the Code 6](#_Toc498292482)

[About the GUI 9](#_Toc498292483)

[User Manual for the Software 9](#_Toc498292484)

[Validation and test cases: 14](#_Toc498292485)

[Symmetrical Bending (Moment acting in 1 direction) 14](#_Toc498292486)

[Symmetrical Bending (Moment acting in 2 directions) 22](#_Toc498292487)

[Limitations of the software 29](#_Toc498292488)

[References: 30](#_Toc498292489)

[List of Figures: 31](#_Toc498292490)

[Appendix 32](#_Toc498292491)

[Code for Test Function 32](#_Toc498292492)

[Code for Polygeon Function 34](#_Toc498292493)

[Code used for GUI 36](#_Toc498292494)

[Code for input gui Function 36](#_Toc498292495)

[Code for page\_2\_gui Function (Page 2 of GUI) 48](#_Toc498292496)

[Code for page\_3\_gui Function (Page 3 in GUI) 53](#_Toc498292497)

[Code for Stress\_point Function 60](#_Toc498292498)

[Code for Stress\_calculator Function 62](#_Toc498292499)

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.

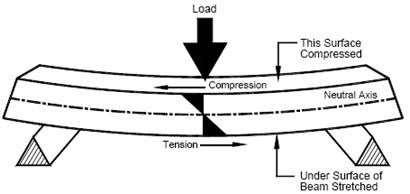


Figure : Compression and Tension in a Beam[2]

1. **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\*R2

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.

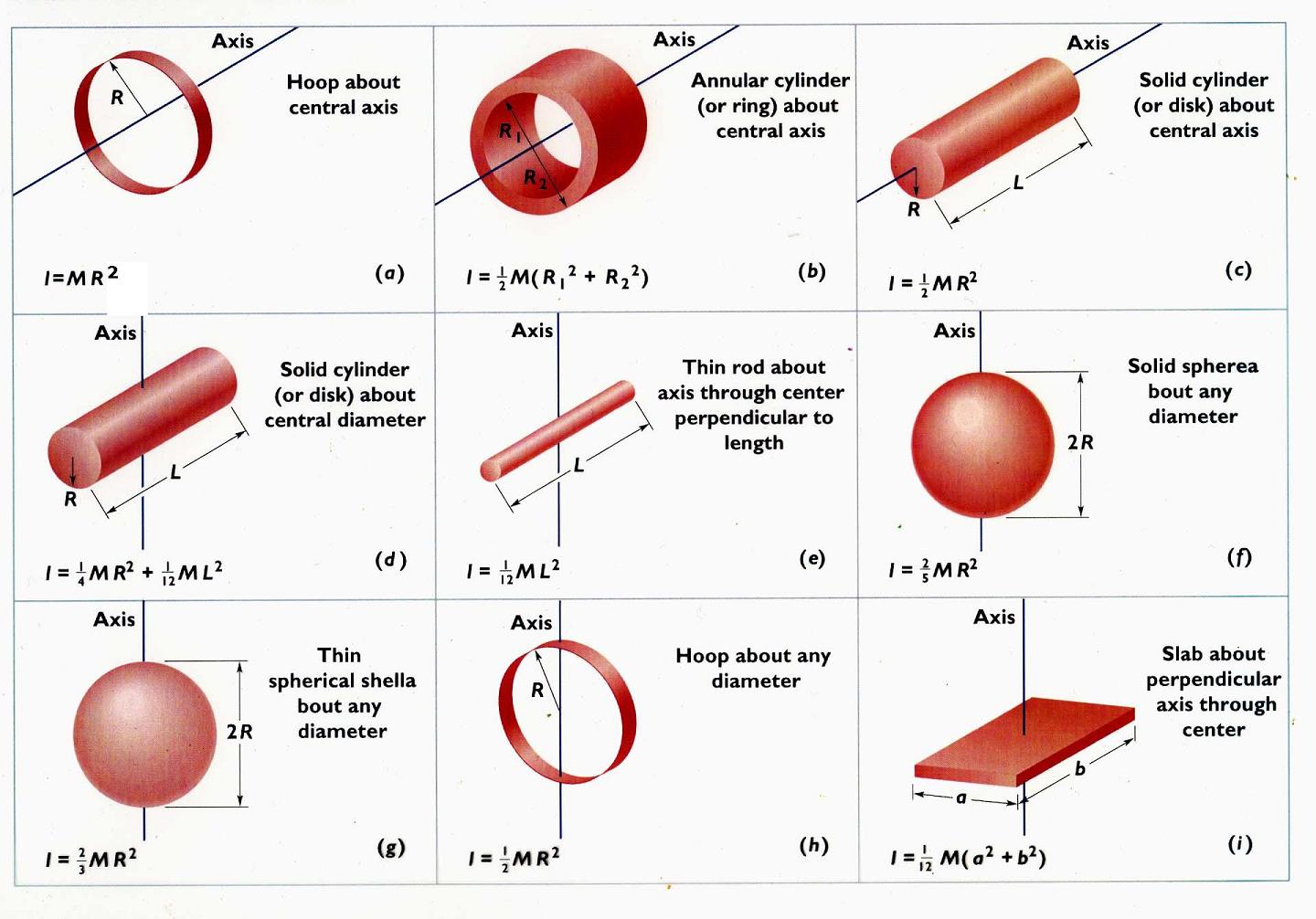
****

Figure : 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.

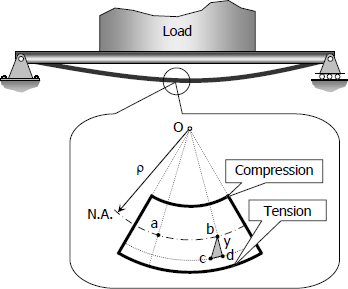


Figure Bending in a beam

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

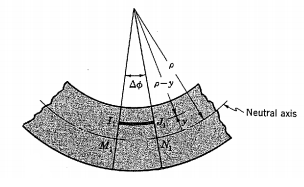


Figure Longitudinal cross section of a beam

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= ρ\*Δϕ

And IJ= (ρ-y)\*Δϕ

Substituting in the equation for Ex we get

Ex = -y/ ρ

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

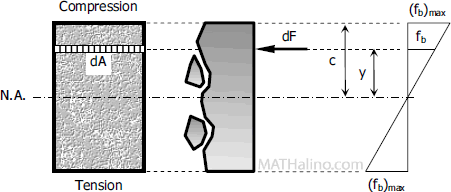


Figure Derivation of stresses in beams

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/ ρ)

dF=dA\*E\*(-y/ ρ)

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/ ρ)

Integrating dM we get

M=-E/ ρ∫(y2\*dA)

∫(y2\*dA)=Ix

M/Ix=-E/ ρ

Multiplying both sides with y we get

M\*y/Ix=E(-y/ ρ)

σ z=My/Ix

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

σ z = My\*x/Iy

σ z = -Mx\*y/Ix

σ z = -Mx\*y/Ix + My\*x/Iy

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

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

σ z=(MyIx + Mx­Ixy)\*x/(Ix Iy – I2yx) + (MxIy + My­Iyx)\*y/(Ix Iy – I2yx)

## 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]:
   1. 
   2. 
   3. 
   4. 
   5. 
   6. 
   7. 
   8. 

Centroidal moments

* 1. 

Principal moments

* 1. 
  2. 

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:

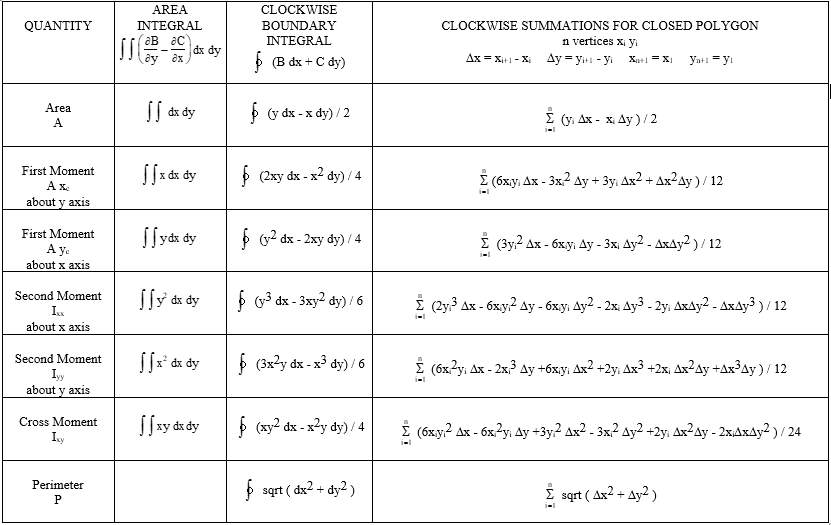


Figure : Green's Theorem and discretization

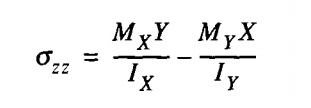
* **Output**

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.



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.

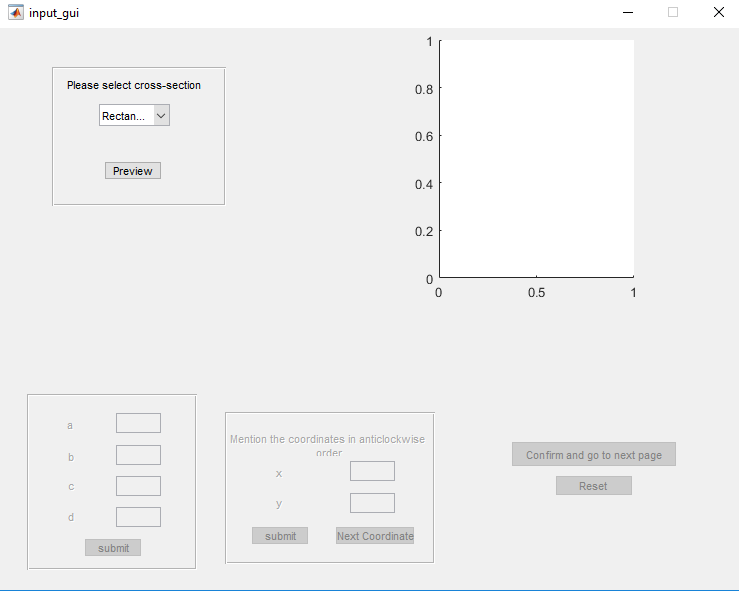


Figure User Manual-Step 1

* **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.

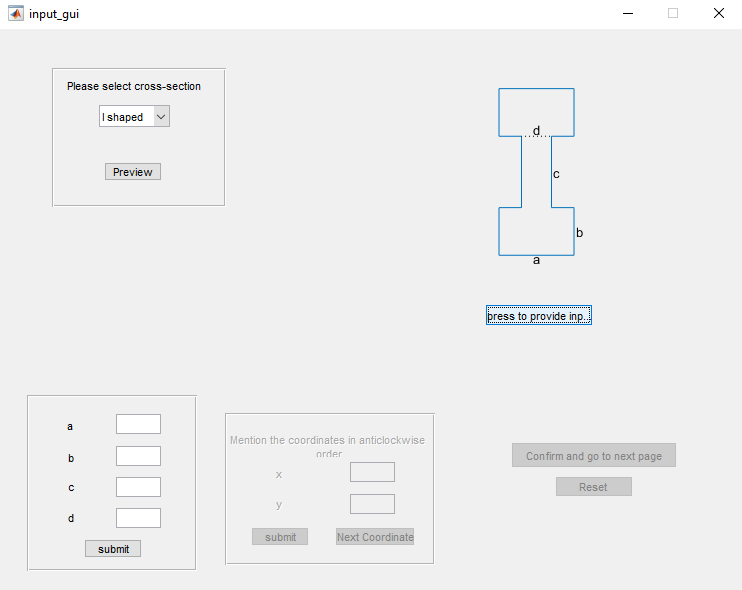


Figure User Manual-Step 2

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.

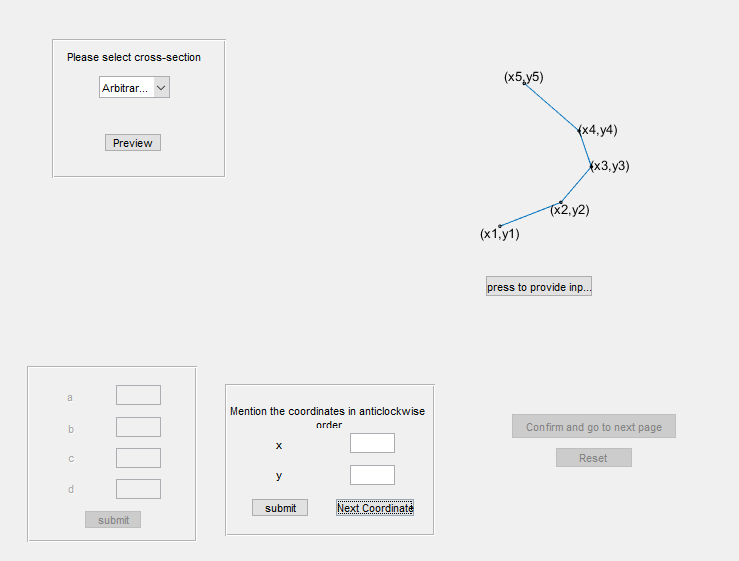


Figure User Manual-Step 2 (Arbitrary)

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 Mx and My.

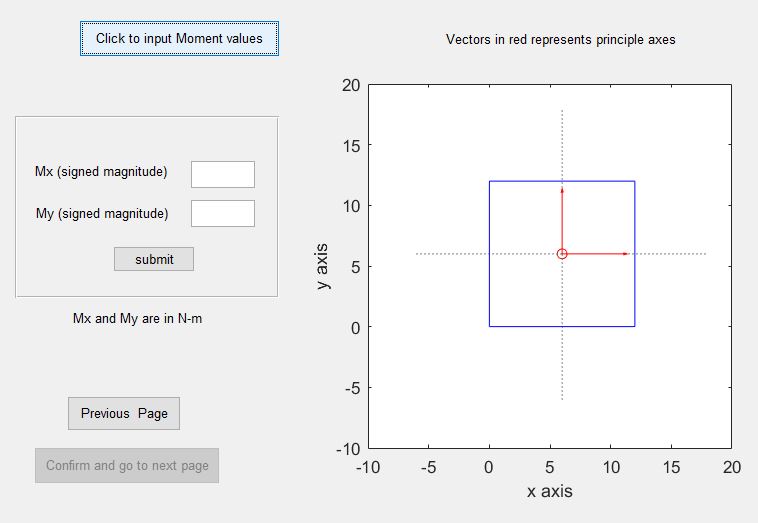


Figure User Manual-Step 3

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

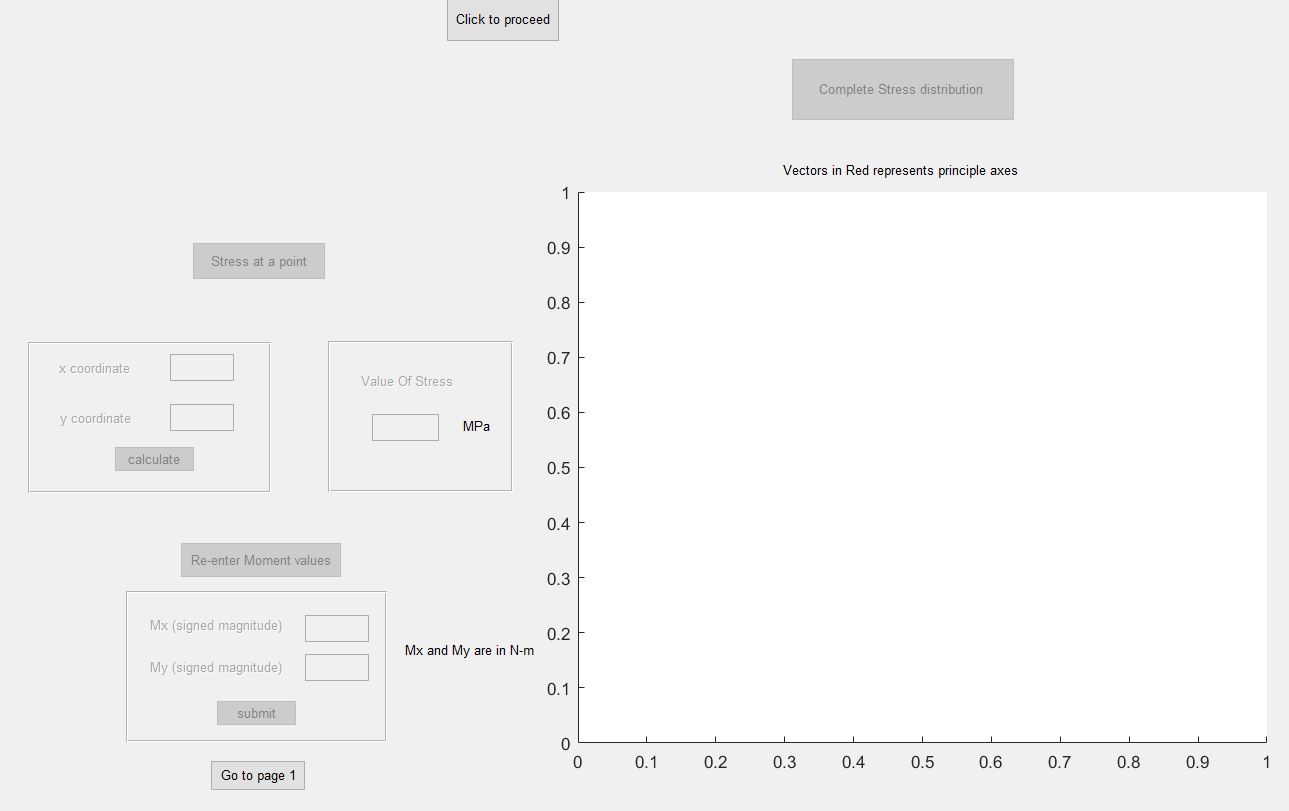


Figure User Manual-Step 4.1

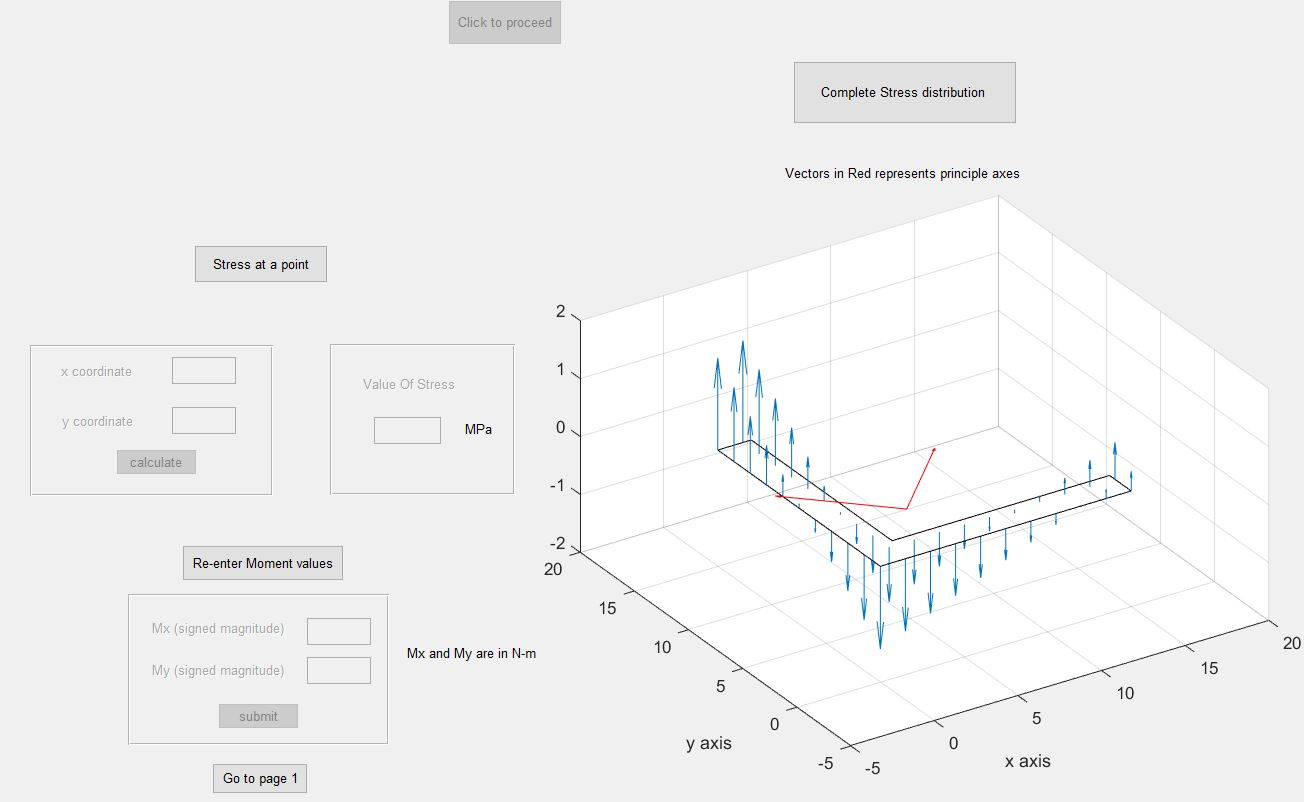


Figure User Manual-Step 4.2

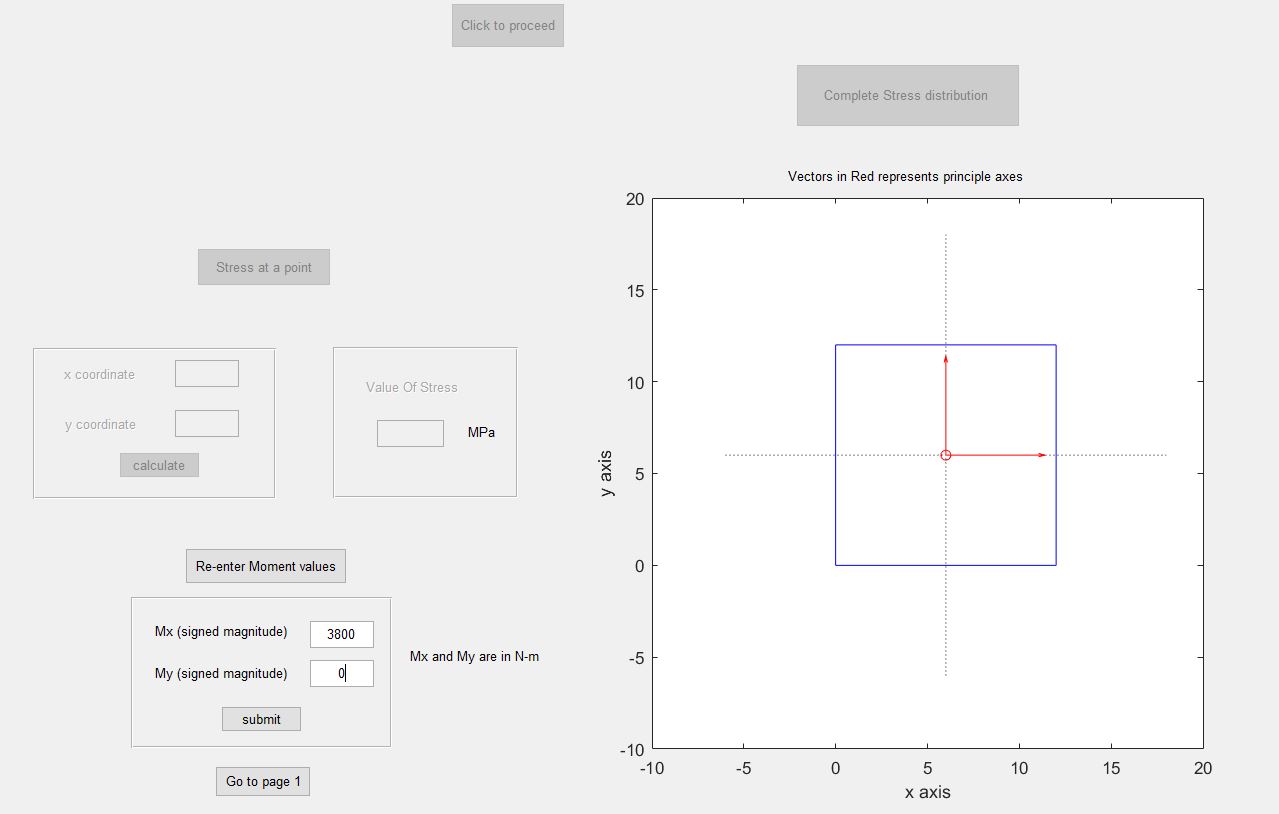


Figure User Manual-Step 4.3

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.

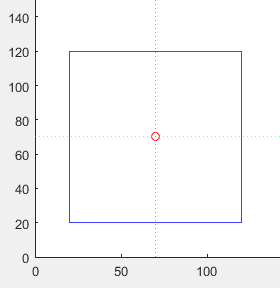


Figure Symmetrical square cross section

`

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 |

1. **Rectangular cross section**

Inputs:

Coordinates: dimensions are in mm.

x = [ 20 80 80 20];

y = [ 20 20 120 120];

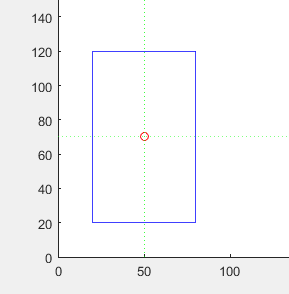


Figure Symmetrical rectangular cross section

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 |

1. **Triangular cross section**

Inputs:

Coordinates: dimensions are in mm.

x = [ 30 90 60];

y = [ 10 10 110];

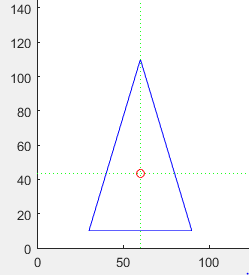


Figure Symmetrical triangular cross section

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 |

1. **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];

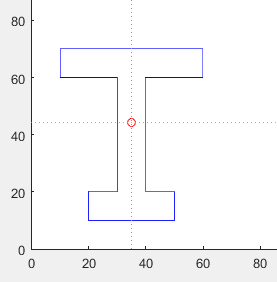


Figure Symmetrical I- cross section

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 |

1. **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];

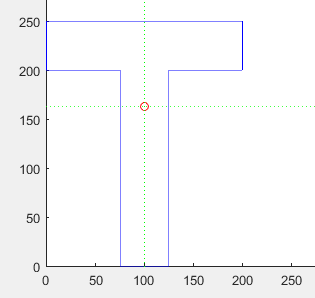


Figure Symmetrical T-cross section

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 |

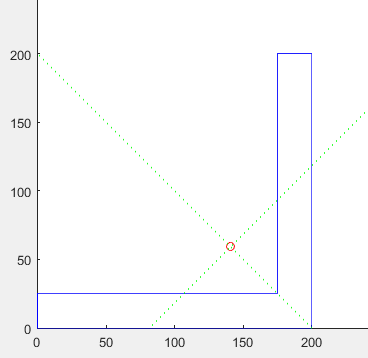
1. **L type cross section**

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.

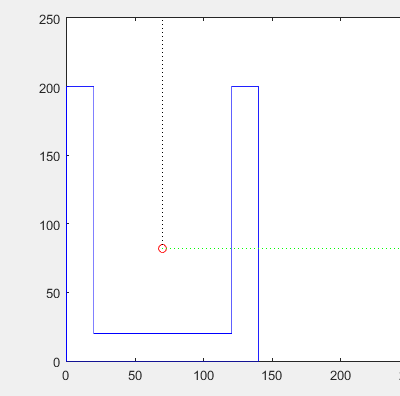
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Moment applied  (kN.m) | Point where moment has to be calculated | | Calculated stress  (MPa) | |
| X component(mm) | Y component(mm) | From code | On paper |
| 10 | 200 | 0 | -412.5 | -413 |
| 10 | 200 | 118.34 | 107.262 | 107.25 |
| 10 | 81 | 0 | -105.513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

1. **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 |

1. **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.

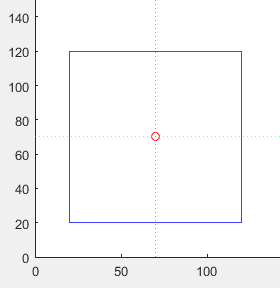


Figure Unsymmetrical square cross section

`

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 |

1. **Rectangular cross section**

Inputs:

Coordinates: dimensions are in mm.

x = [ 20 80 80 20];

y = [ 20 20 120 120];

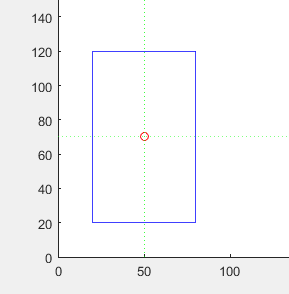


Figure Unsymmetrical rectangular cross section

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 |

1. **Triangular cross section**

Inputs:

Coordinates: dimensions are in mm.

x = [ 30 90 60];

y = [ 10 10 110];

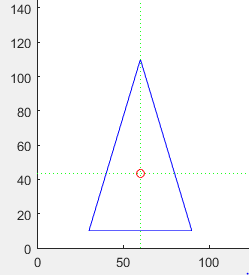


Figure Unsymmetrical triangular cross section

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 |

1. **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];

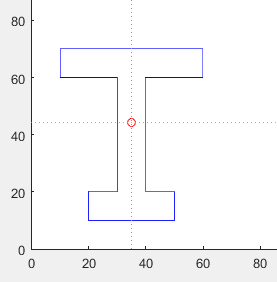


Figure Unsymmetrical I-cross section

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 |

1. **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];

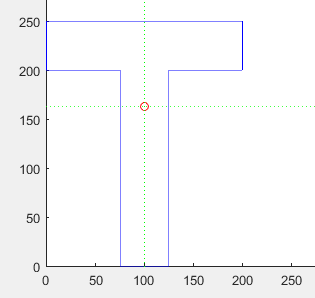


Figure Unsymmetrical T-cross section

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

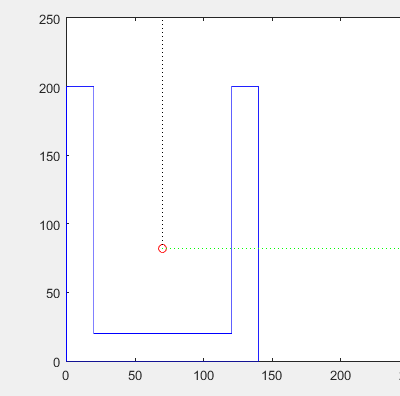
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X direction moment  (kN.m) | Y direction moment  (kN.m) | Point where moment has to be calculated | | Calculated stress  (MPa) | |
| X component(mm) | Y component(mm) | From code | On paper |
| 7.5 | 9.5 | 100 | 0 | -10.7339 | -10.734 |
| 7.5 | 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 |

1. **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](#_Toc498160433)

[Figure 2: Moment of Inertia of some common shapes[4] 3](#_Toc498160434)

[Figure 3 Bending in a beam 4](#_Toc498160435)

[Figure 4 Longitudinal cross section of a beam 4](#_Toc498160436)

[Figure 5 Derivation of stresses in beams 5](#_Toc498160437)

[Figure 6: Green's Theorem and discretization 8](#_Toc498160438)

[Figure 7 User Manual-Step 1 10](#_Toc498160439)

[Figure 8 User Manual-Step 2 10](#_Toc498160440)

[Figure 9 User Manual-Step 2 (Arbitrary) 11](#_Toc498160441)

[Figure 10 User Manual-Step 3 12](#_Toc498160442)

[Figure 11 User Manual-Step 4.1 12](#_Toc498160443)

[Figure 12 User Manual-Step 4.2 13](#_Toc498160444)

[Figure 13 User Manual-Step 4.3 13](#_Toc498160445)

[Figure 14 Square cross section Symmetrical 15](#_Toc498160446)

[Figure 15 Symmetrical rectangular cross section 16](#_Toc498160447)

[Figure 16 Symmetrical triangular cross section 17](#_Toc498160448)

[Figure 17 Symmetrical I- cross section 18](#_Toc498160449)

[Figure 18 Symmetrical T-cross section 19](#_Toc498160450)

[Figure 19 Unsymmetrical square cross section 22](#_Toc498160451)

[Figure 20 Unsymmetrical rectangular cross section 24](#_Toc498160452)

[Figure 21 Unsymmetrical triangular cross section 25](#_Toc498160453)

[Figure 22 Unsymmetrical I-cross section 26](#_Toc498160454)

[Figure 23 Unsymmetrical T-cross section 27](#_Toc498160455)

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.