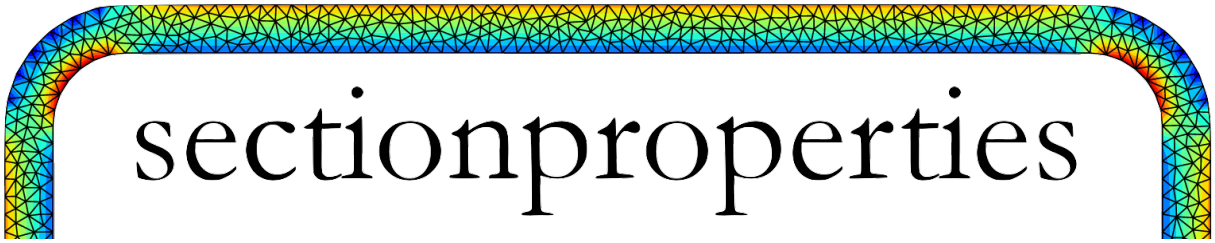
sectionproperties Documentation

Release 1.0.8

Robbie van Leeuwen

Dec 20, 2020

Contents:



sectionproperties is a python package for the analysis of arbitrary cross-sections using the finite element method written by Robbie van Leeuwen. *sectionproperties* can be used to determine section properties to be used in structural design and visualise cross-sectional stresses resulting from combinations of applied forces and bending moments.

A list of the current features of the package and implementation goals for future releases can be found in the README file on github.

CHAPTER 1

Installation

These instructions will get you a copy of *sectionproperties* up and running on your local machine. You will need a working copy of python>=3.5 on your machine.

1.1 Installing *sectionproperties*

sectionproperties uses *meshpy* to efficiently generate a conforming triangular mesh in order to perform a finite element analysis of the structural cross-section. The installation procedure for of *meshpy* depends on your local machine.

1.1.1 UNIX (MacOS/Linux)

sectionproperties and all of its dependencies can be installed through the python package index:

```
$ pip install sectionproperties
```

If you have any issues installing *meshpy*, refer to the installation instructions on its github page or its documentation.

1.1.2 Windows

Install *meshpy* by downloading the appropriate installation wheel.

Navigate to the location of the downloaded wheel and install using pip:

```
$ cd Downloads
$ pip install MeshPy-2018.2.1-cp36-cp36m-win_amd64.whl
```

Once *meshpy* has been installed, the rest of the *sectionproperties* package can be installed using the python package index:

```
$ pip install sectionproperties
```

1.2 Testing the Installation

Python *unittest* modules are located in the *sectionpropertes.tests* package. To see if your installation is working correctly, run this simple test:

```
$ python -m unittest sectionproperties.tests.test_rectangle
```

CHAPTER 2

Structure of an Analysis

The process of performing a cross-section analysis with *sectionproperties* can be broken down into three stages:

1. Pre-Processor: The input geometry and finite element mesh is created.
2. Solver: The cross-section properties are determined.
3. Post-Processor: The results are presented in a number of different formats.

2.1 Creating a Geometry and Mesh

The dimensions and shape of the cross-section to be analysed define the *geometry* of the cross-section. The *sections Module* provides a number of classes to easily generate either commonly used structural sections or an arbitrary cross-section, defined by a list of points, facets and holes. All of the classes in the *sections Module* inherit from the `Geometry` class.

The final stage in the pre-processor involves generating a finite element mesh of the *geometry* that the solver can use to calculate the cross-section properties. This can easily be performed using the `create_mesh()` method that all `Geometry` objects have access to.

The following example creates a geometry object with a circular cross-section. The diameter of the circle is 50 and 64 points are used to discretise the circumference of the circle. A finite element mesh is generated with a maximum triangular area of 2.5:

```
import sectionproperties.pre.sections as sections

geometry = sections.CircularSection(d=50, n=64)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
```

If you are analysing a composite section, or would like to include material properties in your model, material properties can be created using the `Material` class. The following example creates a steel material object:

```
from sectionproperties.pre.pre import Material
```

(continues on next page)

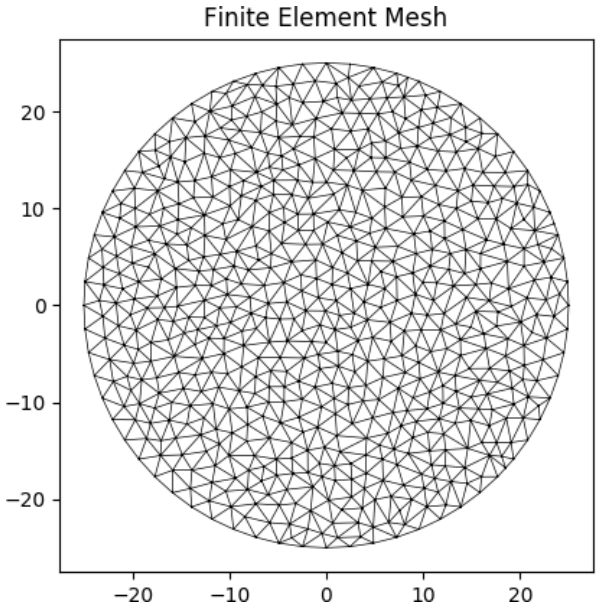


Fig. 1: Finite element mesh generated by the above example.

(continued from previous page)

```
steel = Material(name='Steel', elastic_modulus=200e3, poissons_ratio=0.3, yield_
↪strength=500,
                 color='grey')
```

Refer to *Creating a Geometry, Mesh and Material Properties* for a more detailed explanation of the pre-processing stage.

2.2 Running an Analysis

The solver operates on a *CrossSection* object and can perform four different analysis types:

- Geometric Analysis: calculates area properties.
- Plastic Analysis: calculates plastic properties.
- Warping Analysis: calculates torsion and shear properties.
- Stress Analysis: calculates cross-section stresses.

The geometric analysis can be performed individually. However in order to perform a warping or plastic analysis, a geometric analysis must first be performed. Further, in order to carry out a stress analysis, both a geometric and warping analysis must have already been executed. The program will display a helpful error if you try to run any of these analyses without first performing the prerequisite analyses.

The following example performs a geometric and warping analysis on the circular cross-section defined in the previous section with steel used as the material property:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection
from sectionproperties.pre.pre import Material

geometry = sections.CircularSection(d=50, n=64)
```

(continues on next page)

(continued from previous page)

```
mesh = geometry.create_mesh(mesh_sizes=[2.5])
steel = Material(name='Steel', elastic_modulus=200e3, poissons_ratio=0.3, yield_
↪strength=500,
                 color='grey')

section = CrossSection(geometry, mesh, [steel])
section.calculate_geometric_properties()
section.calculate_warping_properties()
```

Refer to *Running an Analysis* for a more detailed explanation of the solver stage.

2.3 Viewing the Results

Once an analysis has been performed, a number of methods belonging to the *CrossSection* object can be called to present the cross-section results in a number of different formats. For example the cross-section properties can be printed to the terminal, a plot of the centroids displayed and the cross-section stresses visualised in a contour plot.

The following example analyses a 200 PFC section. The cross-section properties are printed to the terminal and a plot of the centroids is displayed:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.PfcSection(d=200, b=75, t_f=12, t_w=6, r=12, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])

section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
section.calculate_warping_properties()

section.plot_centroids()
section.display_results()
```

Output generated by the *display_results()* method:

```
Section Properties:
A       = 2.919699e+03
Qx      = 2.919699e+05
Qy      = 7.122414e+04
cx      = 2.439434e+01
cy      = 1.000000e+02
Ixx_g   = 4.831277e+07
Iyy_g   = 3.392871e+06
Ixy_g   = 7.122414e+06
Ixx_c   = 1.911578e+07
Iyy_c   = 1.655405e+06
Ixy_c   = -6.519258e-09
Zxx+    = 1.911578e+05
Zxx-    = 1.911578e+05
Zyy+    = 3.271186e+04
Zyy-    = 6.786020e+04
rx      = 8.091461e+01
ry      = 2.381130e+01
```

(continues on next page)

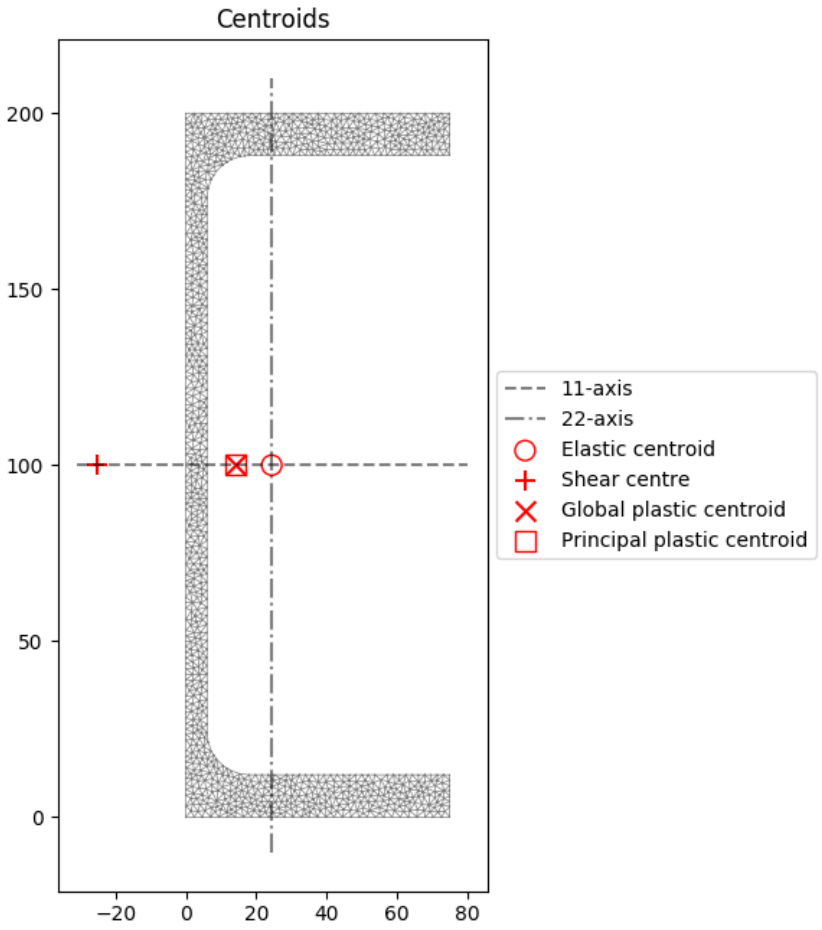


Fig. 2: Plot of the elastic centroid and shear centre for the above example generated by `plot_centroids()`

(continued from previous page)

```
phi      = 0.000000e+00
I11_c    = 1.911578e+07
I22_c    = 1.655405e+06
Z11+     = 1.911578e+05
Z11-     = 1.911578e+05
Z22+     = 3.271186e+04
Z22-     = 6.786020e+04
r11      = 8.091461e+01
r22      = 2.381130e+01
J        = 1.011522e+05
Iw       = 1.039437e+10
x_se     = -2.505109e+01
y_se     = 1.000000e+02
x_st     = -2.505109e+01
y_st     = 1.000000e+02
x1_se    = -4.944543e+01
y2_se    = 4.905074e-06
A_sx     = 9.468851e+02
A_sy     = 1.106943e+03
x_pc     = 1.425046e+01
y_pc     = 1.000000e+02
Sxx      = 2.210956e+05
Syy      = 5.895923e+04
SF_xx+   = 1.156613e+00
SF_xx-   = 1.156613e+00
SF_yy+   = 1.802381e+00
SF_yy-   = 8.688337e-01
x11_pc   = 1.425046e+01
y22_pc   = 1.000000e+02
S11      = 2.210956e+05
S22      = 5.895923e+04
SF_11+   = 1.156613e+00
SF_11-   = 1.156613e+00
SF_22+   = 1.802381e+00
SF_22-   = 8.688337e-01
```

Refer to *Viewing the Results* for a more detailed explanation of the post-processing stage.

CHAPTER 3

Creating a Geometry, Mesh and Material Properties

Before performing a cross-section analysis, the geometry of the cross-section and a finite element mesh must be created. Optionally, material properties can be applied to different regions of the cross-section.

3.1 Cross-Section Geometry

The geometry of a cross-section defines its dimensions and shape and involves the creation of a *Geometry* object. This geometry object stores all the information needed to create a finite element mesh.

class sectionproperties.pre.sections.**Geometry**(*control_points*, *shift*)

Parent class for a cross-section geometry input.

Provides an interface for the user to specify the geometry defining a cross-section. A method is provided for generating a triangular mesh, for translating the cross-section by *(x, y)* and for plotting the geometry.

Variables

- **points** (*list[list[float, float]]*) – List of points *(x, y)* defining the vertices of the cross-section
- **facets** (*list[list[int, int]]*) – List of point index pairs *(p1, p2)* defining the edges of the cross-section
- **holes** (*list[list[float, float]]*) – List of points *(x, y)* defining the locations of holes within the cross-section. If there are no holes, provide an empty list [].
- **control_points** (*list[list[float, float]]*) – A list of points *(x, y)* that define different regions of the cross-section. A control point is an arbitrary point within a region enclosed by facets.
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*
- **perimeter** (*list[int]*) – List of facet indices defining the perimeter of the cross-section

Different regions of the geometry can be specified by defining a list of `control_points`, which are located within unique enclosed areas of the geometry. Different regions can be used to specify different mesh sizes and/or different material properties within the structural cross-section. See the *Examples* for some example scripts in which different regions are specified through a list of control points.

3.2 Creating Common Structural Geometries

In order to make your life easier, there are a number of built-in classes that generate typical structural cross-sections that inherit from the *Geometry* class. Note that these classes automatically assign a `control_point` to the geometry object.

3.2.1 Rectangular Section

class sectionproperties.pre.sections.**RectangularSection**(*d, b, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a rectangular section with the bottom left corner at the origin *(0, 0)*, with depth *d* and width *b*.

Parameters

- **d** (*float*) – Depth (y) of the rectangle
- **b** (*float*) – Width (x) of the rectangle
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a rectangular cross-section with a depth of 100 and width of 50, and generates a mesh with a maximum triangular area of 5:

```
import sectionproperties.pre.sections as sections

geometry = sections.RectangularSection(d=100, b=50)
mesh = geometry.create_mesh(mesh_sizes=[5])
```

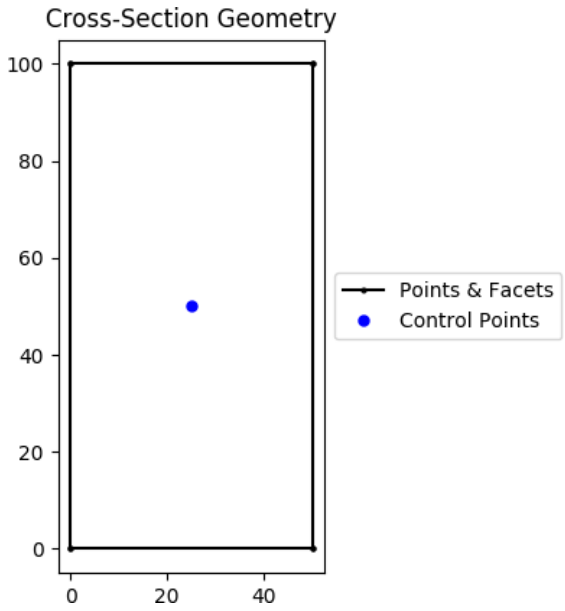


Fig. 1: Rectangular section geometry.

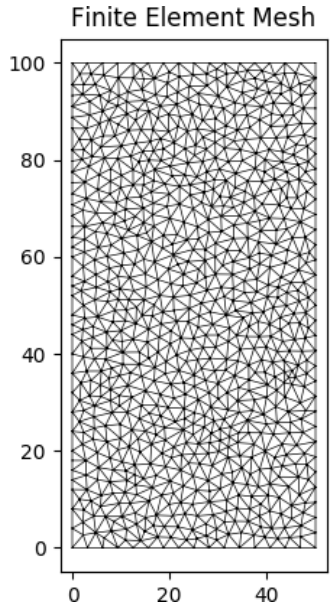


Fig. 2: Mesh generated from the above geometry.

3.2.2 Circular Section

class sectionproperties.pre.sections.**CircularSection**(*d, n, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a solid circle centered at the origin *(0, 0)* with diameter *d* and using *n* points to construct the circle.

Parameters

- **d** (*float*) – Diameter of the circle
- **n** (*int*) – Number of points discretising the circle
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a circular cross-section with a diameter of 50 with 64 points, and generates a mesh with a maximum triangular area of 2.5:

```
import sectionproperties.pre.sections as sections

geometry = sections.CircularSection(d=50, n=64)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
```

3.2.3 Circular Hollow Section (CHS)

class sectionproperties.pre.sections.**Chs**(*d, t, n, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a circular hollow section centered at the origin *(0, 0)*, with diameter *d* and thickness *t*, using *n* points to construct the inner and outer circles.

Parameters

- **d** (*float*) – Outer diameter of the CHS
- **t** (*float*) – Thickness of the CHS

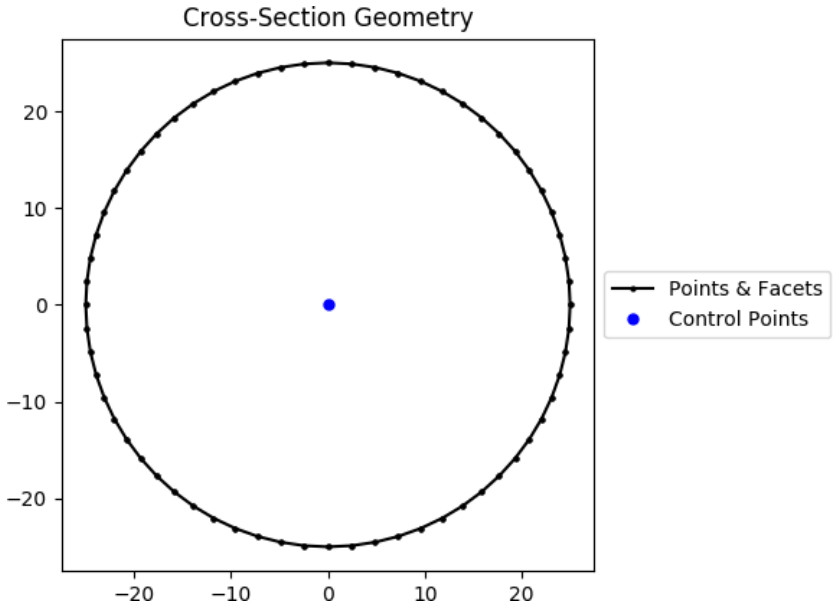


Fig. 3: Circular section geometry.

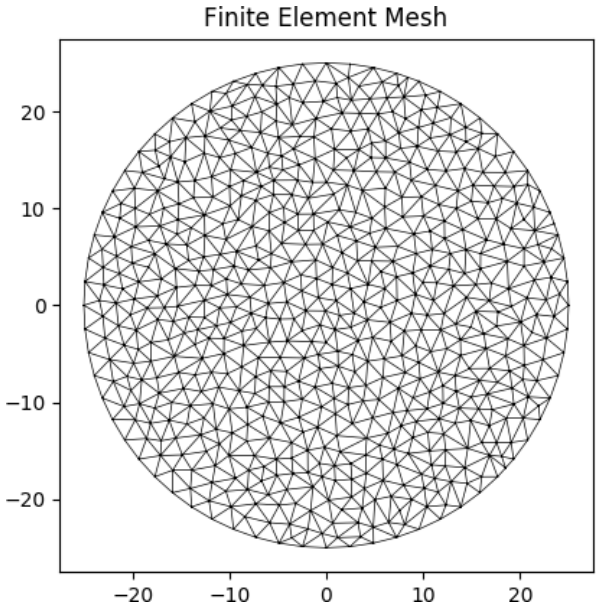


Fig. 4: Mesh generated from the above geometry.

- **n** (*int*) – Number of points discretising the inner and outer circles
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a CHS discretised with 64 points, with a diameter of 48 and thickness of 3.2, and generates a mesh with a maximum triangular area of 1.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.Chs(d=48, t=3.2, n=64)
mesh = geometry.create_mesh(mesh_sizes=[1.0])
```

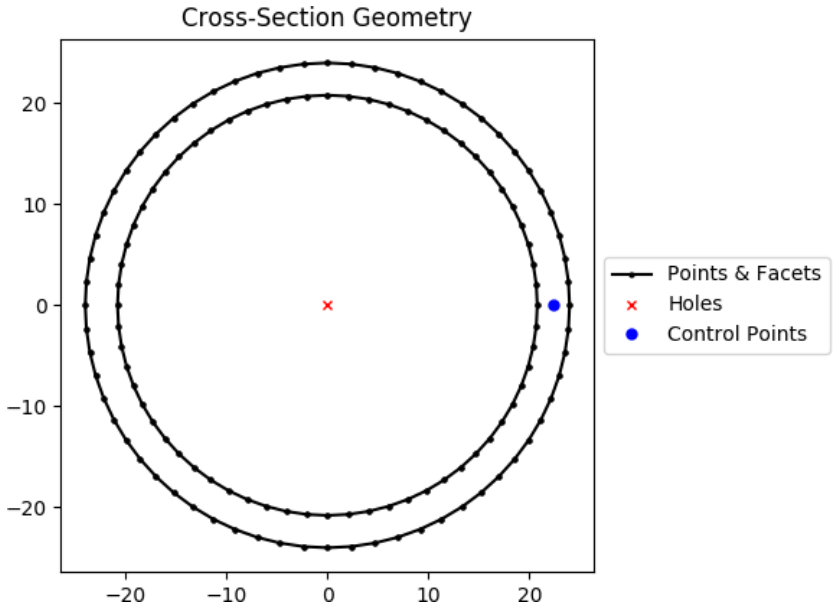


Fig. 5: CHS geometry.

3.2.4 Elliptical Section

class sectionproperties.pre.sections.**EllipticalSection**(*d_y, d_x, n, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a solid ellipse centered at the origin *(0, 0)* with vertical diameter *d_y* and horizontal diameter *d_x*, using *n* points to construct the ellipse.

Parameters

- **d_y** (*float*) – Diameter of the ellipse in the y-dimension
- **d_x** (*float*) – Diameter of the ellipse in the x-dimension
- **n** (*int*) – Number of points discretising the ellipse
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates an elliptical cross-section with a vertical diameter of 25 and horizontal diameter of 50, with 40 points, and generates a mesh with a maximum triangular area of 1.0:

```
import sectionproperties.pre.sections as sections
```

(continues on next page)

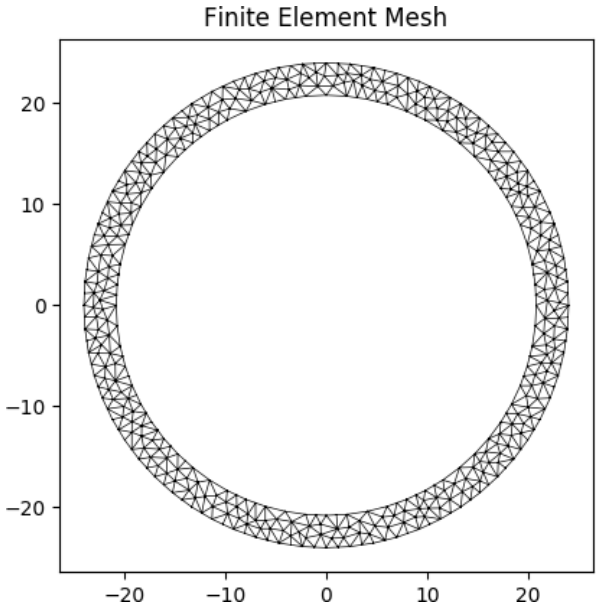


Fig. 6: Mesh generated from the above geometry.

(continued from previous page)

```
geometry = sections.EllipticalSection(d_y=25, d_x=50, n=40)
mesh = geometry.create_mesh(mesh_sizes=[1.0])
```

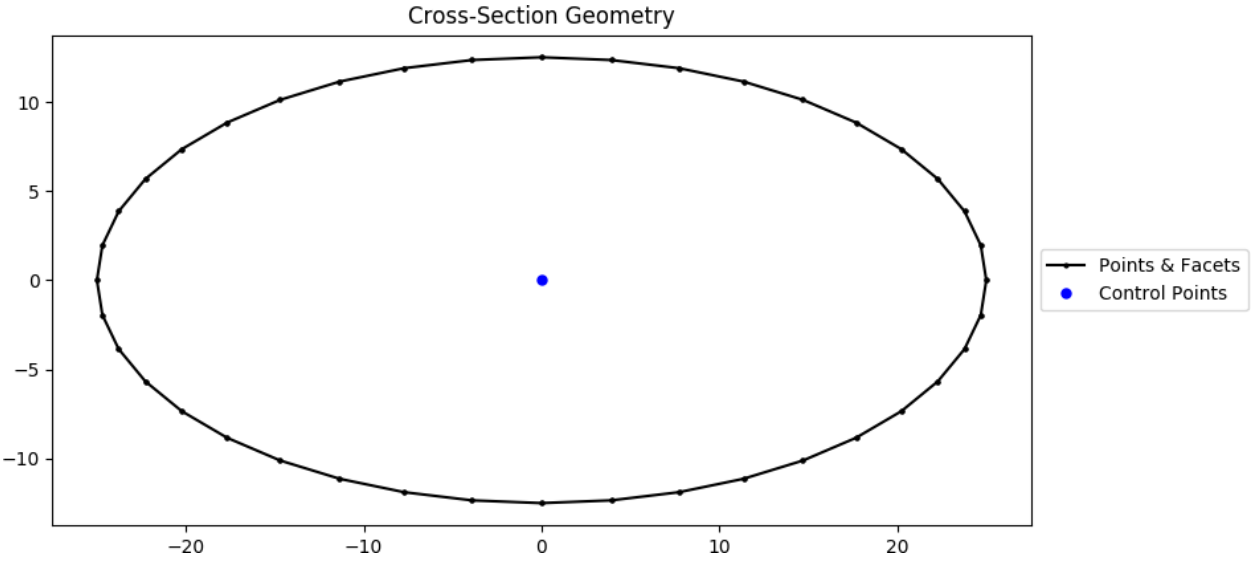


Fig. 7: Elliptical section geometry.

3.2.5 Elliptical Hollow Section (EHS)

class sectionproperties.pre.sections.**Ehs**(*d_y, d_x, t, n, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs an elliptical hollow section centered at the origin *(0, 0)*, with outer vertical diameter *d_y*, outer

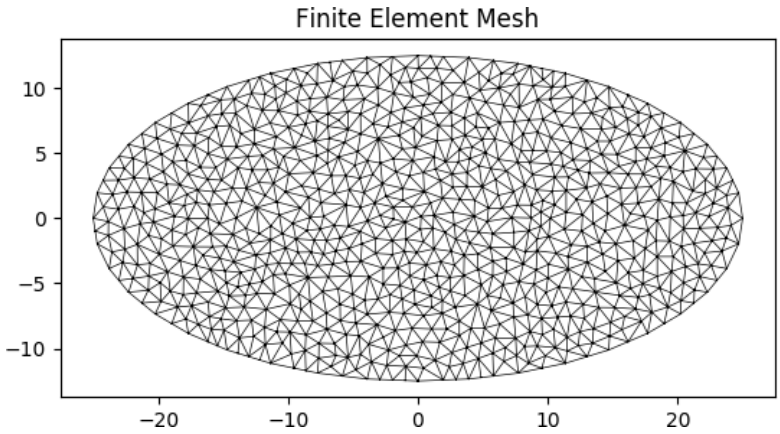


Fig. 8: Mesh generated from the above geometry.

horizontal diameter *d_x*, and thickness *t*, using *n* points to construct the inner and outer ellipses.

Parameters

- **d_y** (*float*) – Diameter of the ellipse in the y-dimension
- **d_x** (*float*) – Diameter of the ellipse in the x-dimension
- **t** (*float*) – Thickness of the EHS
- **n** (*int*) – Number of points discretising the inner and outer ellipses
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a EHS discretised with 30 points, with a outer vertical diameter of 25, outer horizontal diameter of 50, and thickness of 2.0, and generates a mesh with a maximum triangular area of 0.5:

```
import sectionproperties.pre.sections as sections

geometry = sections.Ehs(d_y=25, d_x=50, t=2.0, n=64)
mesh = geometry.create_mesh(mesh_sizes=[0.5])
```

3.2.6 Rectangular Hollow Section (RHS)

class sectionproperties.pre.sections.**Rhs**(*d, b, t, r_out, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a rectangular hollow section centered at *(b/2, d/2)*, with depth *d*, width *b*, thickness *t* and outer radius *r_out*, using *n_r* points to construct the inner and outer radii. If the outer radius is less than the thickness of the RHS, the inner radius is set to zero.

Parameters

- **d** (*float*) – Depth of the RHS
- **b** (*float*) – Width of the RHS
- **t** (*float*) – Thickness of the RHS
- **r_out** (*float*) – Outer radius of the RHS
- **n_r** (*int*) – Number of points discretising the inner and outer radii
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

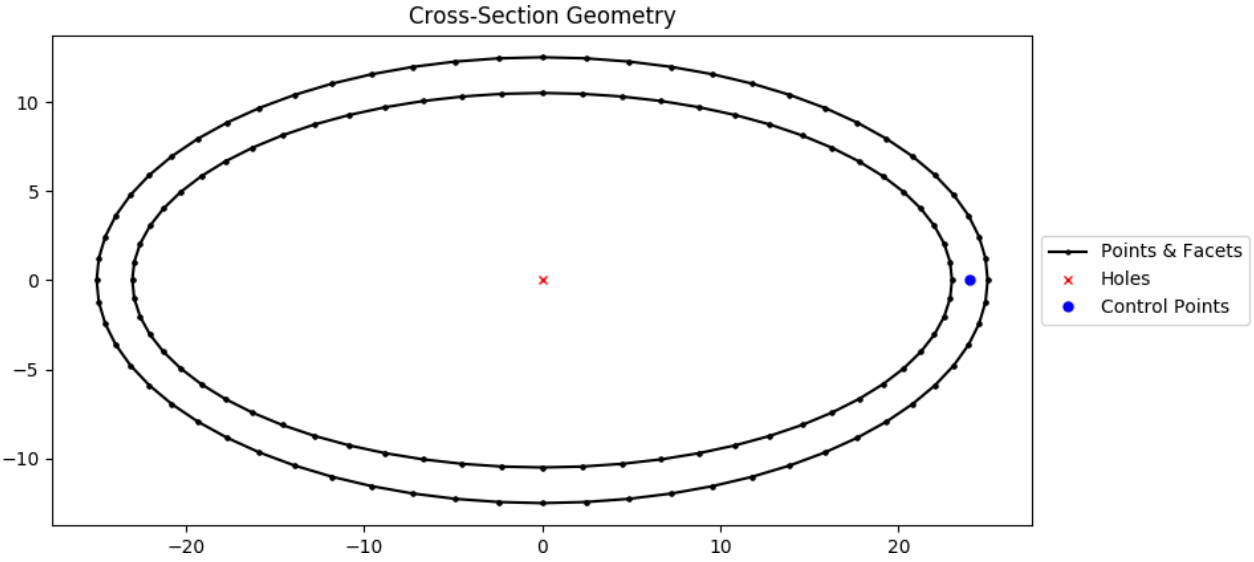


Fig. 9: EHS geometry.

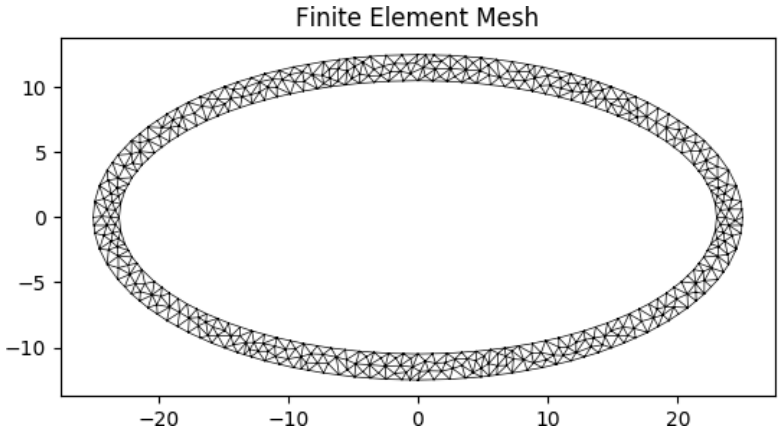


Fig. 10: Mesh generated from the above geometry.

The following example creates an RHS with a depth of 100, a width of 50, a thickness of 6 and an outer radius of 9, using 8 points to discretise the inner and outer radii. A mesh is generated with a maximum triangular area of 2.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.Rhs(d=100, b=50, t=6, r_out=9, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.0])
```

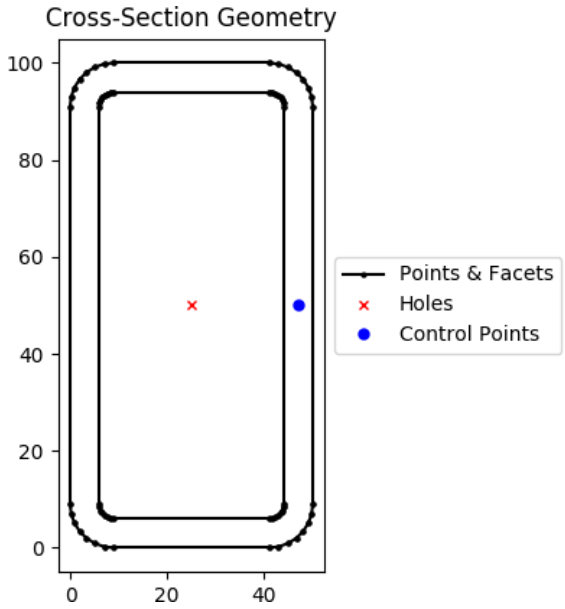


Fig. 11: RHS geometry.

3.2.7 I-Section

class sectionproperties.pre.sections.**ISection**(*d, b, t_f, t_w, r, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs an I-section centered at *(b/2, d/2)*, with depth *d*, width *b*, flange thickness *t_f*, web thickness *t_w*, and root radius *r*, using *n_r* points to construct the root radius.

Parameters

- **d** (*float*) – Depth of the I-section
- **b** (*float*) – Width of the I-section
- **t_f** (*float*) – Flange thickness of the I-section
- **t_w** (*float*) – Web thickness of the I-section
- **r** (*float*) – Root radius of the I-section
- **n_r** (*int*) – Number of points discretising the root radius
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates an I-section with a depth of 203, a width of 133, a flange thickness of 7.8, a web thickness of 5.8 and a root radius of 8.9, using 16 points to discretise the root radius. A mesh is generated with a maximum triangular area of 3.0:

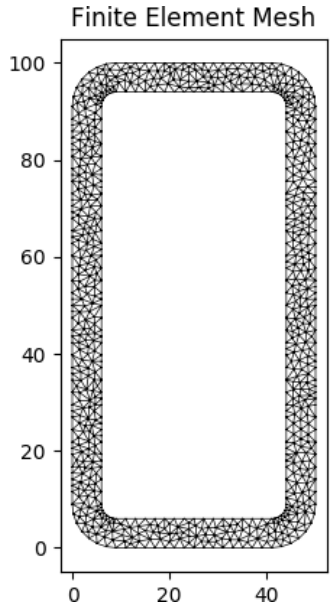


Fig. 12: Mesh generated from the above geometry.

```
import sectionproperties.pre.sections as sections

geometry = sections.ISection(d=203, b=133, t_f=7.8, t_w=5.8, r=8.9, n_
↪r=16)
mesh = geometry.create_mesh(mesh_sizes=[3.0])
```

3.2.8 Monosymmetric I-Section

class sectionproperties.pre.sections.**MonoISection**(*d, b_t, b_b, t_fb, t_ft, t_w, r, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a monosymmetric I-section centered at *(max(b_t, b_b)/2, d/2)*, with depth *d*, top flange width *b_t*, bottom flange width *b_b*, top flange thickness *t_ft*, top flange thickness *t_fb*, web thickness *t_w*, and root radius *r*, using *n_r* points to construct the root radius.

Parameters

- **d** (*float*) – Depth of the I-section
- **b_t** (*float*) – Top flange width
- **b_b** (*float*) – Bottom flange width
- **t_ft** (*float*) – Top flange thickness of the I-section
- **t_fb** (*float*) – Bottom flange thickness of the I-section
- **t_w** (*float*) – Web thickness of the I-section
- **r** (*float*) – Root radius of the I-section
- **n_r** (*int*) – Number of points discretising the root radius
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

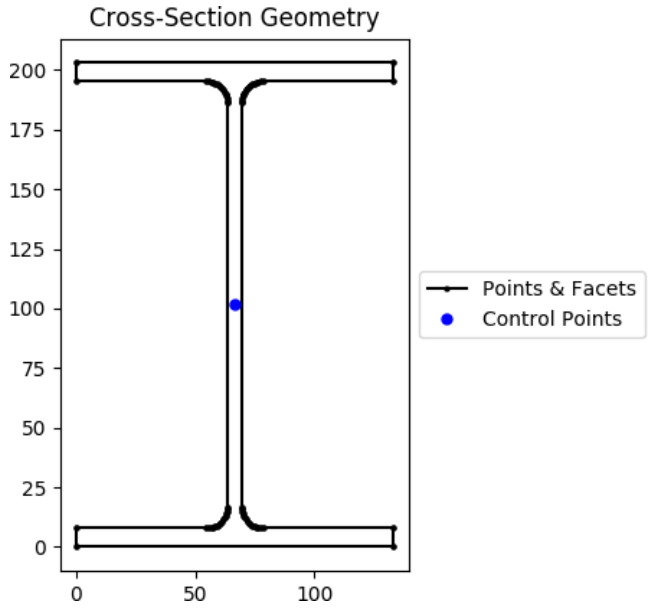


Fig. 13: I-section geometry.

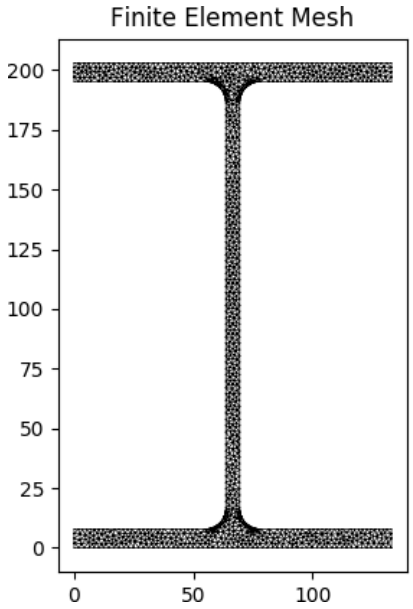


Fig. 14: Mesh generated from the above geometry.

The following example creates a monosymmetric I-section with a depth of 200, a top flange width of 50, a top flange thickness of 12, a bottom flange width of 130, a bottom flange thickness of 8, a web thickness of 6 and a root radius of 8, using 16 points to discretise the root radius. A mesh is generated with a maximum triangular area of 3.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.MonoISection(
    d=200, b_t=50, b_b=130, t_ft=12, t_fb=8, t_w=6, r=8, n_r=16
)
mesh = geometry.create_mesh(mesh_sizes=[3.0])
```

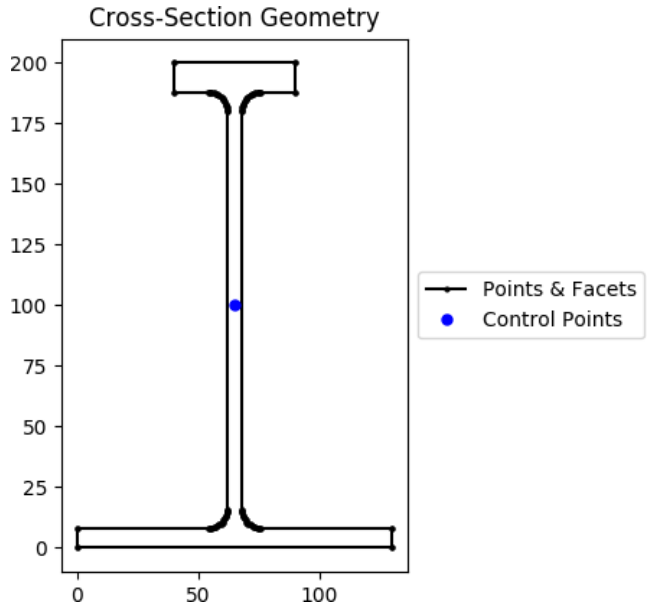


Fig. 15: I-section geometry.

3.2.9 Tapered Flange I-Section

class sectionproperties.pre.sections.**TaperedFlangeISection**(*d, b, t_f, t_w, r_r, r_f, alpha, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a Tapered Flange I-section centered at *(b/2, d/2)*, with depth *d*, width *b*, mid-flange thickness *t_f*, web thickness *t_w*, root radius *r_r*, flange radius *r_f* and flange angle *alpha*, using *n_r* points to construct the radii.

Parameters

- **d** (*float*) – Depth of the Tapered Flange I-section
- **b** (*float*) – Width of the Tapered Flange I-section
- **t_f** (*float*) – Mid-flange thickness of the Tapered Flange I-section (measured at the point equidistant from the face of the web to the edge of the flange)

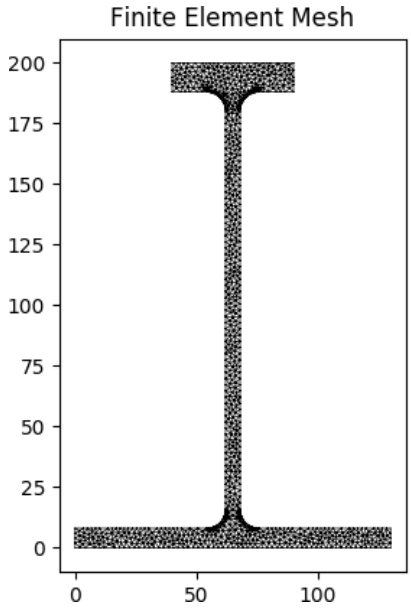


Fig. 16: Mesh generated from the above geometry.

- **t_w** (*float*) – Web thickness of the Tapered Flange I-section
- **r_r** (*float*) – Root radius of the Tapered Flange I-section
- **r_f** (*float*) – Flange radius of the Tapered Flange I-section
- **alpha** (*float*) – Flange angle of the Tapered Flange I-section (degrees)
- **n_r** (*int*) – Number of points discretising the radii
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a Tapered Flange I-section with a depth of 588, a width of 191, a mid-flange thickness of 27.2, a web thickness of 15.2, a root radius of 17.8, a flange radius of 8.9 and a flange angle of 8°, using 16 points to discretise the radii. A mesh is generated with a maximum triangular area of 20.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.TaperedFlangeISection(
    d=588, b=191, t_f=27.2, t_w=15.2, r_r=17.8, r_f=8.9, alpha=8, n_r=16
)
mesh = geometry.create_mesh(mesh_sizes=[20.0])
```

3.2.10 Parallel Flange Channel (PFC) Section

class sectionproperties.pre.sections.**PfcSection**(*d, b, t_f, t_w, r, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a PFC section with the bottom left corner at the origin *(0, 0)*, with depth *d*, width *b*, flange thickness *t_f*, web thickness *t_w* and root radius *r*, using *n_r* points to construct the root radius.

Parameters

- **d** (*float*) – Depth of the PFC section

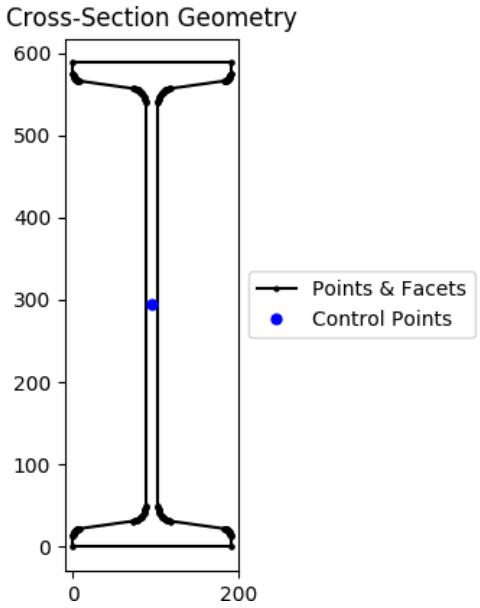


Fig. 17: I-section geometry.

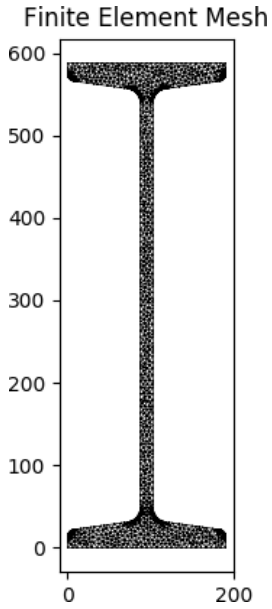


Fig. 18: Mesh generated from the above geometry.

- **b** (*float*) – Width of the PFC section
- **t_f** (*float*) – Flange thickness of the PFC section
- **t_w** (*float*) – Web thickness of the PFC section
- **r** (*float*) – Root radius of the PFC section
- **n_r** (*int*) – Number of points discretising the root radius
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a PFC section with a depth of 250, a width of 90, a flange thickness of 15, a web thickness of 8 and a root radius of 12, using 8 points to discretise the root radius. A mesh is generated with a maximum triangular area of 5.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.PfcSection(d=250, b=90, t_f=15, t_w=8, r=12, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[5.0])
```

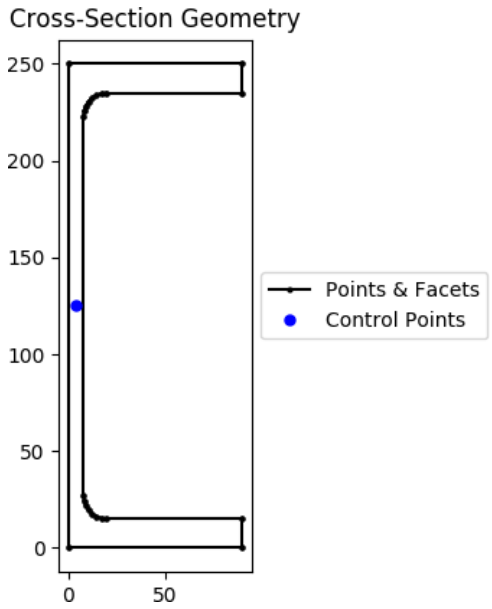


Fig. 19: PFC geometry.

3.2.11 Tapered Flange Channel Section

class sectionproperties.pre.sections.**TaperedFlangeChannel**(*d, b, t_f, t_w, r_r, r_f, alpha, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a Tapered Flange Channel section with the bottom left corner at the origin *(0, 0)*, with depth *d*, width *b*, mid-flange thickness *t_f*, web thickness *t_w*, root radius *r_r*, flange radius *r_f* and flange angle *alpha*, using *n_r* points to construct the radii.

Parameters

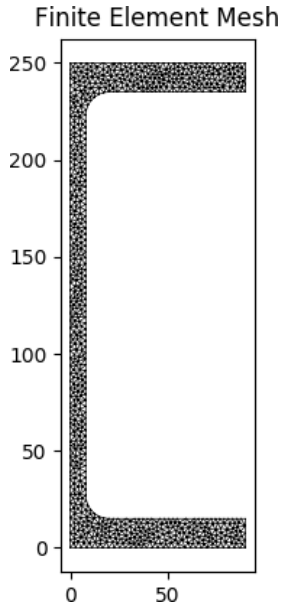


Fig. 20: Mesh generated from the above geometry.

- **d** (*float*) – Depth of the Tapered Flange Channel section
- **b** (*float*) – Width of the Tapered Flange Channel section
- **t_f** (*float*) – Mid-flange thickness of the Tapered Flange Channel section (measured at the point equidistant from the face of the web to the edge of the flange)
- **t_w** (*float*) – Web thickness of the Tapered Flange Channel section
- **r_r** (*float*) – Root radius of the Tapered Flange Channel section
- **r_f** (*float*) – Flange radius of the Tapered Flange Channel section
- **alpha** (*float*) – Flange angle of the Tapered Flange Channel section (degrees)
- **n_r** (*int*) – Number of points discretising the radii
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a Tapered Flange Channel section with a depth of 10, a width of 3.5, a mid-flange thickness of 0.575, a web thickness of 0.475, a root radius of 0.575, a flange radius of 0.4 and a flange angle of 8°, using 16 points to discretise the radii. A mesh is generated with a maximum triangular area of 0.02:

```
import sectionproperties.pre.sections as sections

geometry = sections.TaperedFlangeChannel(
    d=10, b=3.5, t_f=0.575, t_w=0.475, r_r=0.575, r_f=0.4, alpha=8, n_
↪r=16
)
mesh = geometry.create_mesh(mesh_sizes=[0.02])
```

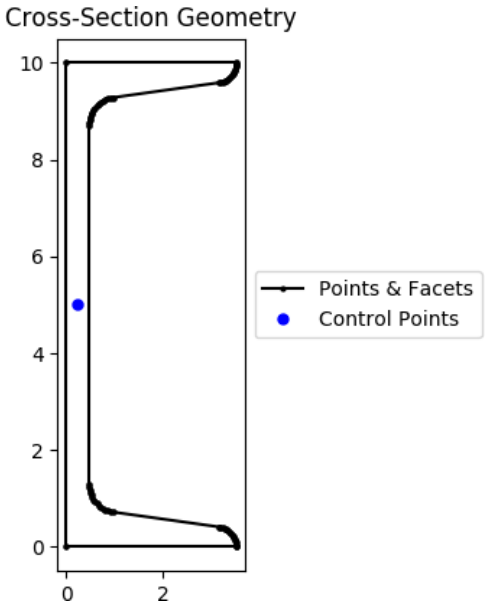


Fig. 21: I-section geometry.

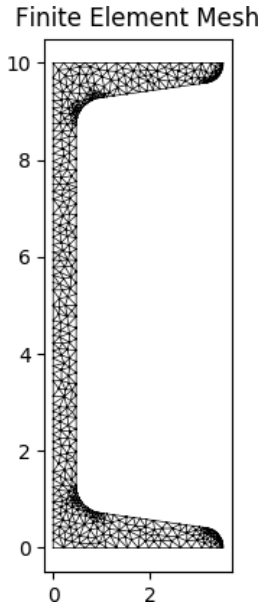


Fig. 22: Mesh generated from the above geometry.

3.2.12 Tee Section

class sectionproperties.pre.sections.**TeeSection**(*d, b, t_f, t_w, r, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a Tee section with the top left corner at *(0, d)*, with depth *d*, width *b*, flange thickness *t_f*, web thickness *t_w* and root radius *r*, using *n_r* points to construct the root radius.

Parameters

- **d** (*float*) – Depth of the Tee section
- **b** (*float*) – Width of the Tee section
- **t_f** (*float*) – Flange thickness of the Tee section
- **t_w** (*float*) – Web thickness of the Tee section
- **r** (*float*) – Root radius of the Tee section
- **n_r** (*int*) – Number of points discretising the root radius
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a Tee section with a depth of 200, a width of 100, a flange thickness of 12, a web thickness of 6 and a root radius of 8, using 8 points to discretise the root radius. A mesh is generated with a maximum triangular area of 3.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.TeeSection(d=200, b=100, t_f=12, t_w=6, r=8, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[3.0])
```

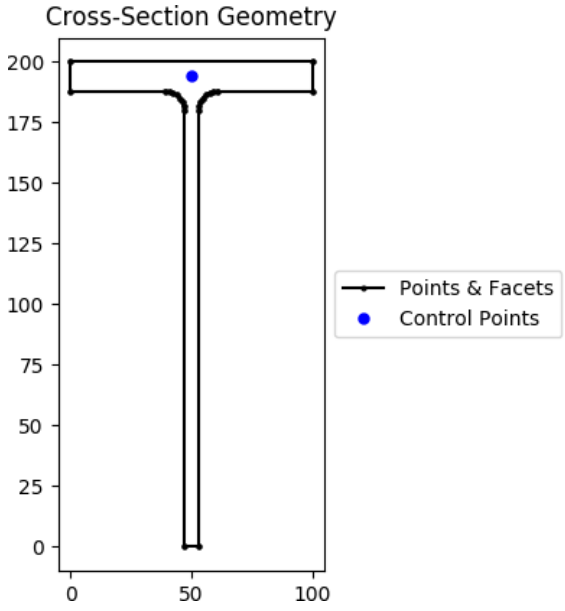


Fig. 23: Tee section geometry.

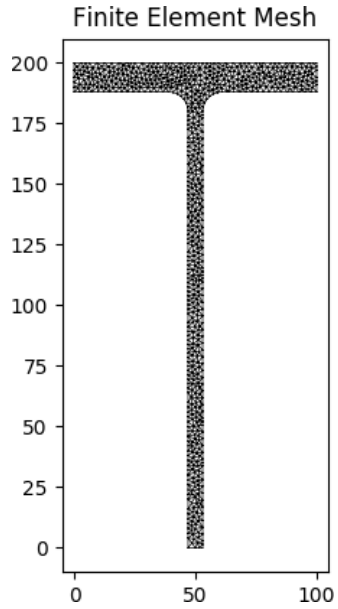


Fig. 24: Mesh generated from the above geometry.

3.2.13 Angle Section

class sectionproperties.pre.sections.**AngleSection**(*d, b, t, r_r, r_t, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs an angle section with the bottom left corner at the origin *(0, 0)*, with depth *d*, width *b*, thickness *t*, root radius *r_r* and toe radius *r_t*, using *n_r* points to construct the radii.

Parameters

- **d** (*float*) – Depth of the angle section
- **b** (*float*) – Width of the angle section
- **t** (*float*) – Thickness of the angle section
- **r_r** (*float*) – Root radius of the angle section
- **r_t** (*float*) – Toe radius of the angle section
- **n_r** (*int*) – Number of points discretising the radii
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates an angle section with a depth of 150, a width of 100, a thickness of 8, a root radius of 12 and a toe radius of 5, using 16 points to discretise the radii. A mesh is generated with a maximum triangular area of 2.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.AngleSection(d=150, b=100, t=8, r_r=12, r_t=5, n_
↪r=16)
mesh = geometry.create_mesh(mesh_sizes=[2.0])
```

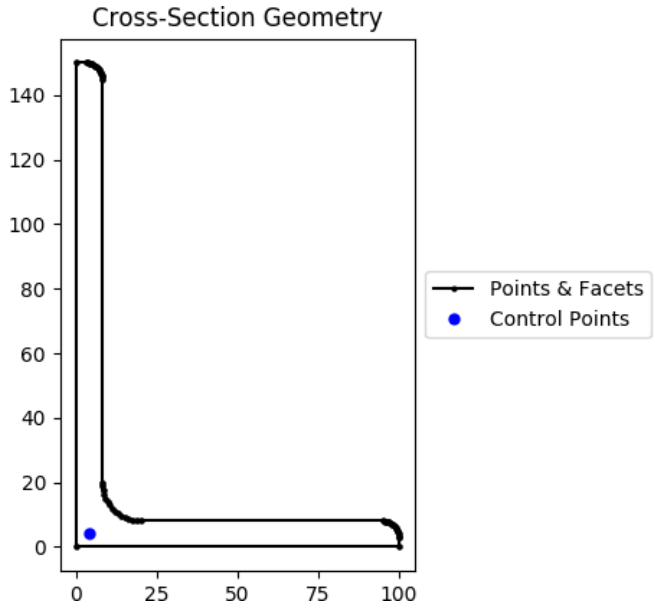
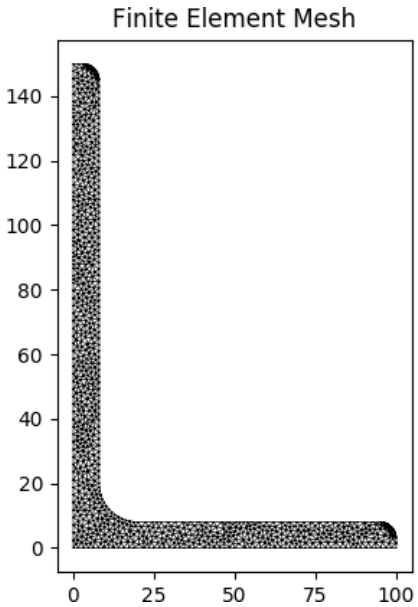


Fig. 25: Angle section geometry.



3.2.14 Cee Section

class sectionproperties.pre.sections.**CeeSection**(*d, b, l, t, r_out, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a Cee section with the bottom left corner at the origin *(0, 0)*, with depth *d*, width *b*, lip *l*, thickness *t* and outer radius *r_out*, using *n_r* points to construct the radius. If the outer radius is less than the thickness of the Cee Section, the inner radius is set to zero.

Parameters

- **d** (*float*) – Depth of the Cee section
- **b** (*float*) – Width of the Cee section
- **l** (*float*) – Lip of the Cee section
- **t** (*float*) – Thickness of the Cee section
- **r_out** (*float*) – Outer radius of the Cee section
- **n_r** (*int*) – Number of points discretising the outer radius
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

Raises **Exception** – Lip length must be greater than the outer radius

The following example creates a Cee section with a depth of 125, a width of 50, a lip of 30, a thickness of 1.5 and an outer radius of 6, using 8 points to discretise the radius. A mesh is generated with a maximum triangular area of 0.25:

```
import sectionproperties.pre.sections as sections

geometry = sections.CeeSection(d=125, b=50, l=30, t=1.5, r_out=6, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[0.25])
```

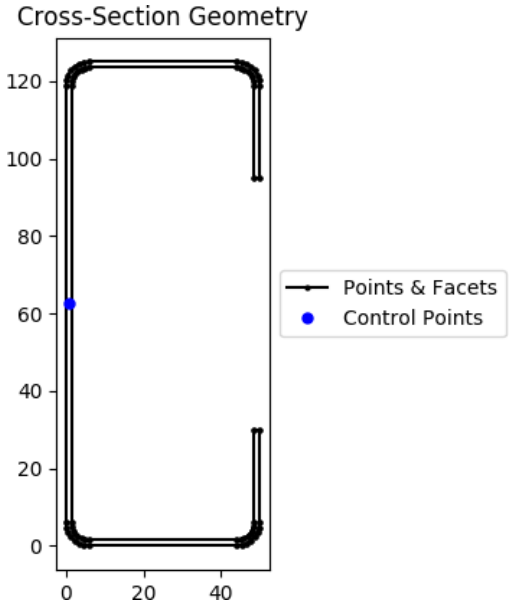
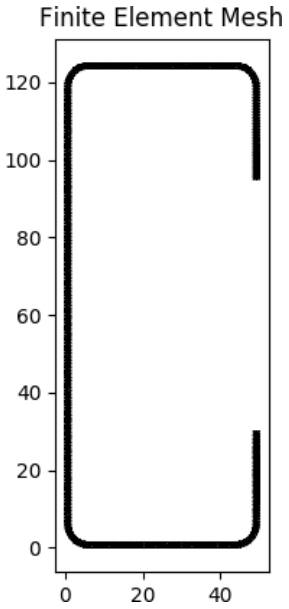


Fig. 26: Cee section geometry.



3.2.15 Zed Section

class sectionproperties.pre.sections.**ZedSection**(*d, b_l, b_r, l, t, r_out, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a Zed section with the bottom left corner at the origin *(0, 0)*, with depth *d*, left flange width *b_l*, right flange width *b_r*, lip *l*, thickness *t* and outer radius *r_out*, using *n_r* points to construct the radius. If the outer radius is less than the thickness of the Zed Section, the inner radius is set to zero.

Parameters

- **d** (*float*) – Depth of the Zed section
- **b_l** (*float*) – Left flange width of the Zed section
- **b_r** (*float*) – Right flange width of the Zed section
- **l** (*float*) – Lip of the Zed section
- **t** (*float*) – Thickness of the Zed section
- **r_out** (*float*) – Outer radius of the Zed section
- **n_r** (*int*) – Number of points discretising the outer radius
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

Raises **Exception** – Lip length must be greater than the outer radius

The following example creates a Zed section with a depth of 100, a left flange width of 40, a right flange width of 50, a lip of 20, a thickness of 1.2 and an outer radius of 5, using 8 points to discretise the radius. A mesh is generated with a maximum triangular area of 0.15:

```
import sectionproperties.pre.sections as sections

geometry = sections.ZedSection(d=100, b_l=40, b_r=50, l=20, t=1.2, r_
↪out=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[0.15])
```

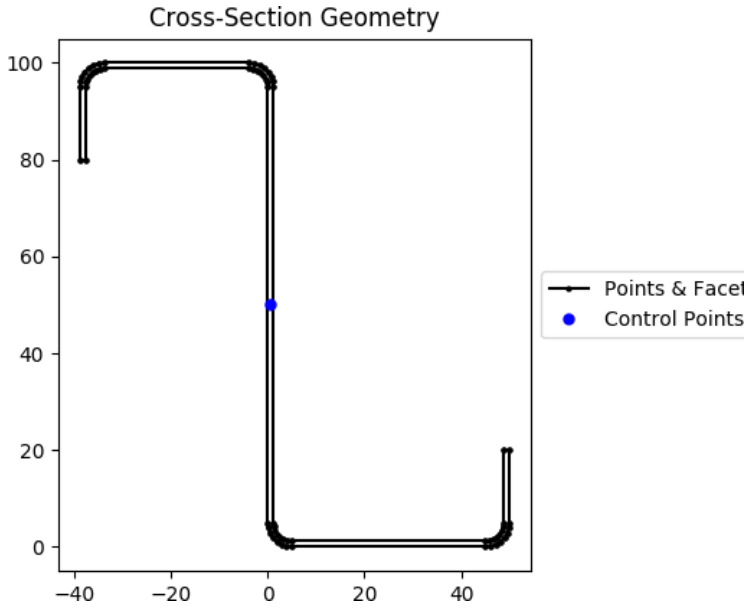
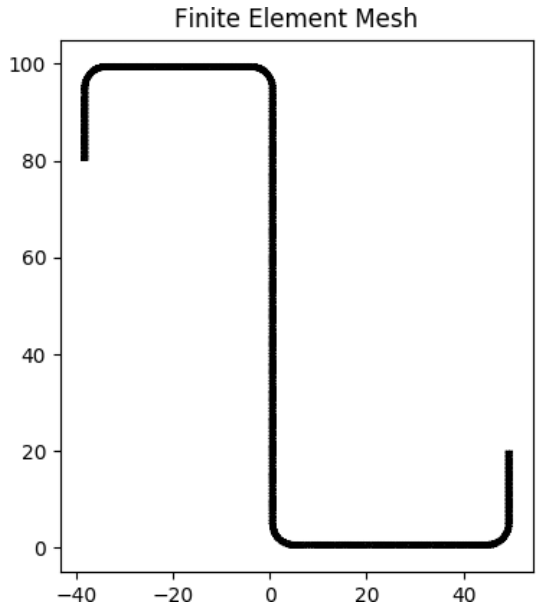


Fig. 27: Zed section geometry.



3.2.16 Cruciform Section

class sectionproperties.pre.sections.**CruciformSection**(*d, b, t, r, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a cruciform section centered at the origin *(0, 0)*, with depth *d*, width *b*, thickness *t* and root radius *r*, using *n_r* points to construct the root radius.

Parameters

- **d** (*float*) – Depth of the cruciform section
- **b** (*float*) – Width of the cruciform section
- **t** (*float*) – Thickness of the cruciform section
- **r** (*float*) – Root radius of the cruciform section
- **n_r** (*int*) – Number of points discretising the root radius
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a cruciform section with a depth of 250, a width of 175, a thickness of 12 and a root radius of 16, using 16 points to discretise the radius. A mesh is generated with a maximum triangular area of 5.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.CruciformSection(d=250, b=175, t=12, r=16, n_r=16)
mesh = geometry.create_mesh(mesh_sizes=[5.0])
```

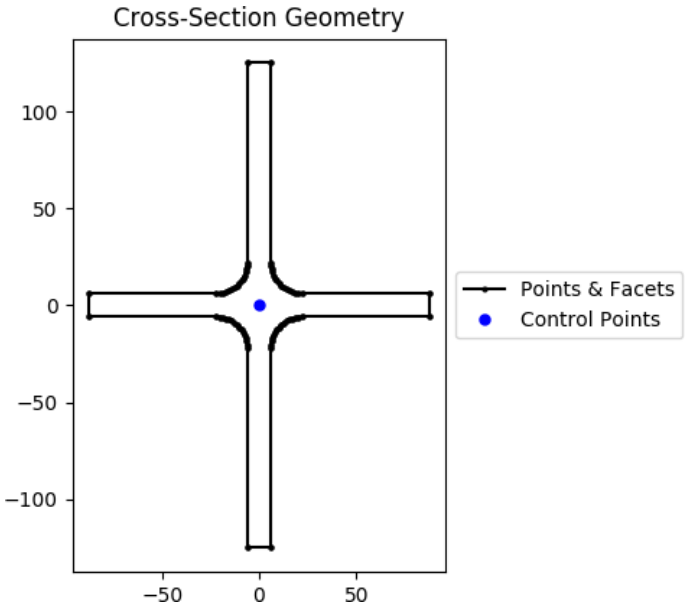
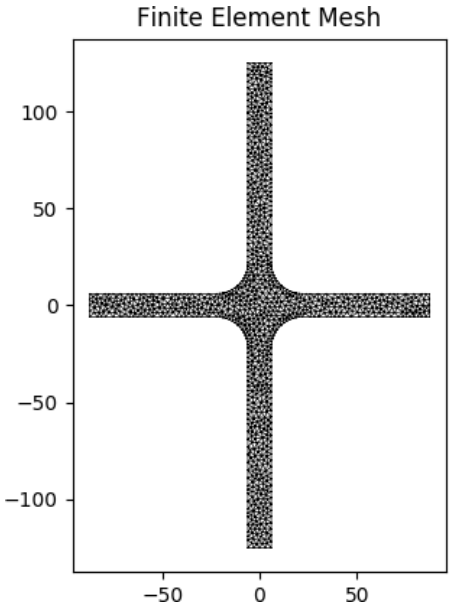


Fig. 28: Cruciform section geometry.



3.2.17 Polygon Section

class sectionproperties.pre.sections.**PolygonSection**(*d, t, n_sides, r_in=0, n_r=1, rot=0, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a regular hollow polygon section centered at *(0, 0)*, with a pitch circle diameter of bounding polygon *d*, thickness *t*, number of sides *n_sides* and an optional inner radius *r_in*, using *n_r* points to construct the inner and outer radii (if radii is specified).

Parameters

- **d** (*float*) – Pitch circle diameter of the outer bounding polygon (i.e. diameter of circle that passes through all vertices of the outer polygon)
- **t** (*float*) – Thickness of the polygon section wall
- **r_in** (*float*) – Inner radius of the polygon corners. By default, if not specified, a polygon with no corner radii is generated.
- **n_r** (*int*) – Number of points discretising the inner and outer radii, ignored if no inner radii is specified
- **rot** – Initial counterclockwise rotation in degrees. By default bottom face is aligned with x axis.
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

Raises **Exception** – Number of sides in polygon must be greater than or equal to 3

The following example creates an Octagonal section (8 sides) with a diameter of 200, a thickness of 6 and an inner radius of 20, using 12 points to discretise the inner and outer radii. A mesh is generated with a maximum triangular area of 5:

```
import sectionproperties.pre.sections as sections
```

(continues on next page)

(continued from previous page)

```
geometry = sections.PolygonSection(d=200, t=6, n_sides=8, r_in=20, n_
↪r=12)
mesh = geometry.create_mesh(mesh_sizes=[5])
```

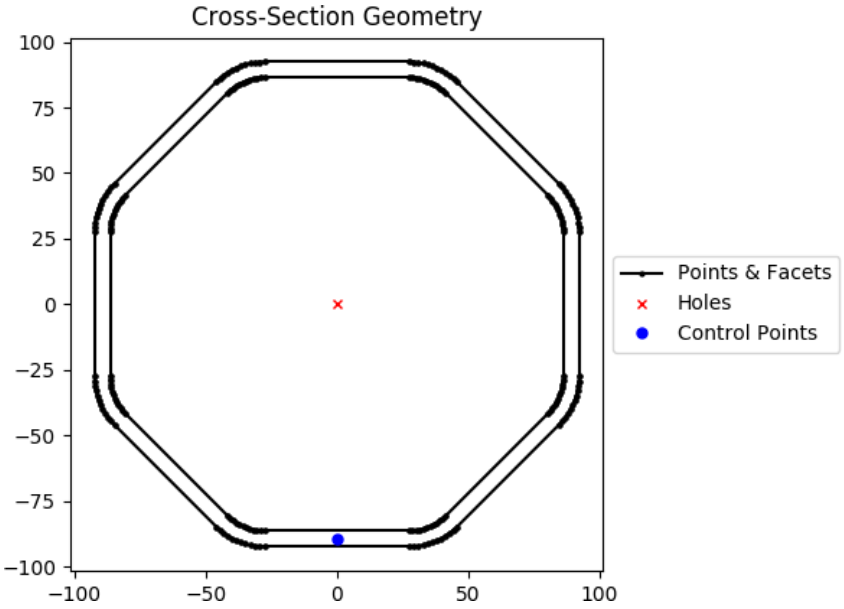


Fig. 29: Octagonal section geometry.

3.2.18 Box Girder Section

class sectionproperties.pre.sections.**BoxGirderSection**(*d*, *b_t*, *b_b*, *t_ft*, *t_fb*, *t_w*, *shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a Box Girder section centered at at *(max(b_t, b_b)/2, d/2)*, with depth *d*, top width *b_t*, bottom width *b_b*, top flange thickness *t_ft*, bottom flange thickness *t_fb* and web thickness *t_w*.

Parameters

- **d** (*float*) – Depth of the Box Girder section
- **b_t** (*float*) – Top width of the Box Girder section
- **b_b** (*float*) – Bottom width of the Box Girder section
- **t_ft** (*float*) – Top lange thickness of the Box Girder section
- **t_fb** (*float*) – Bottom flange thickness of the Box Girder section
- **t_w** (*float*) – Web thickness of the Box Girder section
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a Box Gider section with a depth of 1200, a top width of 1200, a bottom width of 400, a top flange thickness of 16, a bottom flange thickness of 12 and a web thickness of 8. A mesh is generated with a maximum triangular area of 5.0:

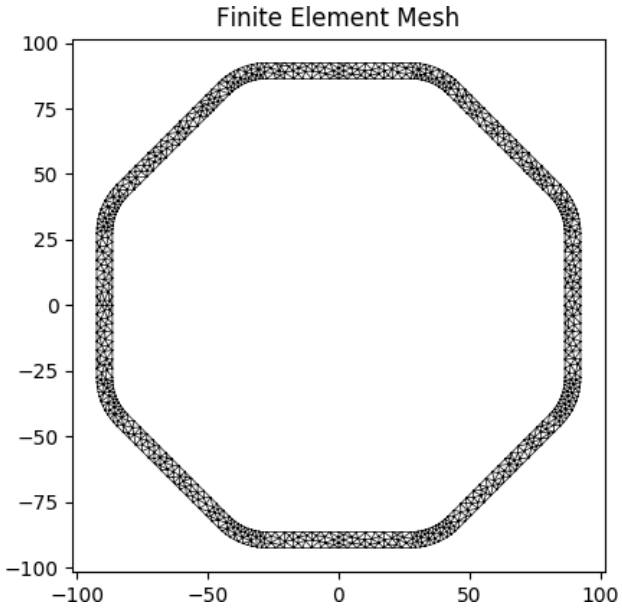


Fig. 30: Mesh generated from the above geometry.

```
import sectionproperties.pre.sections as sections

geometry = sections.BoxGirderSection(d=1200, b_t=1200, b_b=400, t_ft=100,
↪ t_fb=80, t_w=50)
mesh = geometry.create_mesh(mesh_sizes=[200.0])
```

3.3 Arbitrary Cross-Section Geometries

If none of the above classes gives you what you need, you can create a *CustomSection* geometry object, which is defined by a list of points (nodes), facets (node connectivities) and hole locations:

class sectionproperties.pre.sections.**CustomSection**(*points, facets, holes, control_points, shift=[0, 0], perimeter=[]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a cross-section from a list of points, facets, holes and a user specified control point.

Parameters

- **points** (*list[list[float, float]]*) – List of points *(x, y)* defining the vertices of the cross-section
- **facets** (*list[list[int, int]]*) – List of point index pairs *(p1, p2)* defining the edges of the cross-section
- **holes** (*list[list[float, float]]*) – List of points *(x, y)* defining the locations of holes within the cross-section. If there are no holes, provide an empty list [].
- **control_points** (*list[list[float, float]]*) – A list of points *(x, y)* that define different regions of the cross-section. A control point is an arbitrary point within a region enclosed by facets.
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

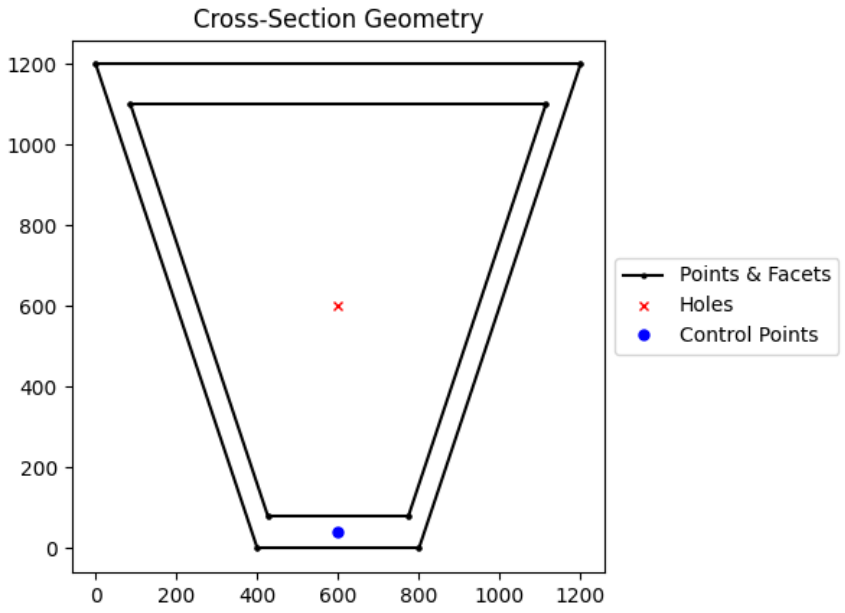


Fig. 31: Box Girder geometry.

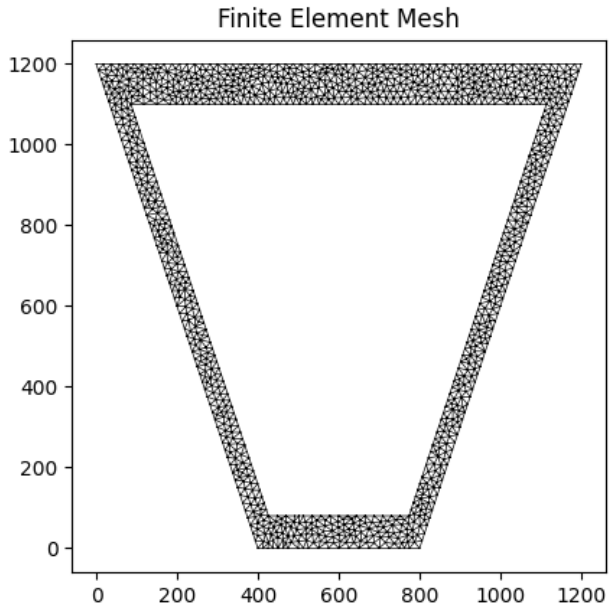


Fig. 32: Mesh generated from the above geometry.

- **perimeter** – List of facet indices defining the perimeter of the cross-section

The following example creates a hollow trapezium with a base width of 100, top width of 50, height of 50 and a wall thickness of 10. A mesh is generated with a maximum triangular area of 2.0:

```
import sectionproperties.pre.sections as sections

points = [[0, 0], [100, 0], [75, 50], [25, 50], [15, 10], [85, 10], [70, 40], [30,
↪ 40]]
facets = [[0, 1], [1, 2], [2, 3], [3, 0], [4, 5], [5, 6], [6, 7], [7, 4]]
holes = [[50, 25]]
control_points = [[5, 5]]
perimeter = [0, 1, 2, 3]

geometry = sections.CustomSection(
    points, facets, holes, control_points, perimeter=perimeter
)
mesh = geometry.create_mesh(mesh_sizes=[2.0])
```

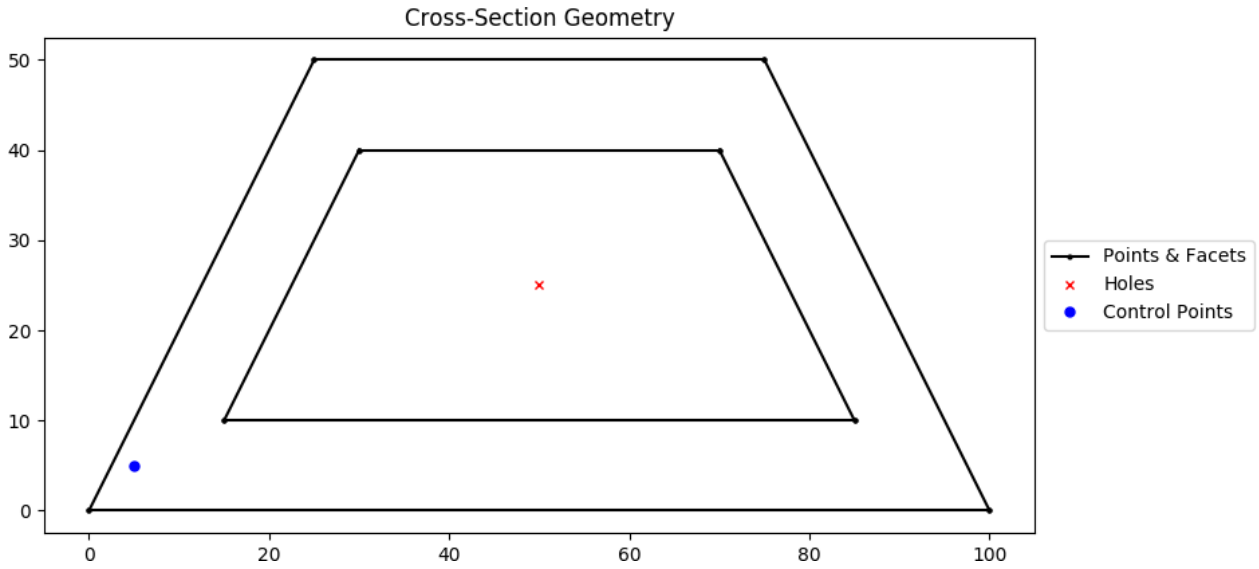


Fig. 33: Custom section geometry.

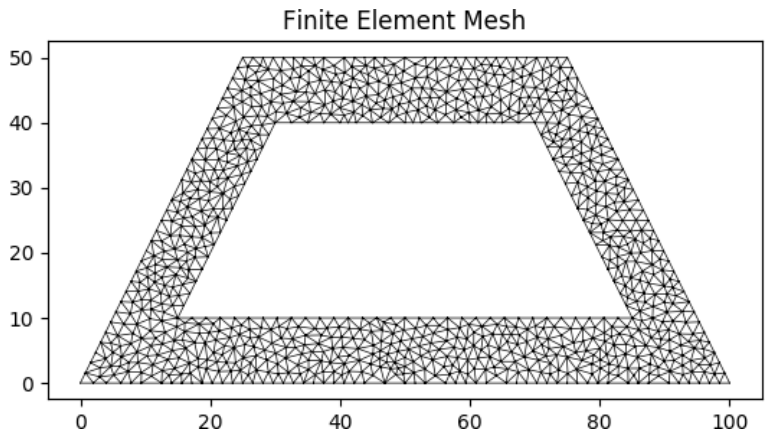


Fig. 34: Mesh generated from the above geometry.

Note: Ensure that the `control_points` you choose lie within the *CustomSection*. If any of the `control_points` are outside the region, on an edge or within a hole, the meshing algorithm will likely not treat distinct areas within the *CustomSection* as a separate regions and mesh refinements may not work as anticipated.

Note: In order to calculate the perimeter of the cross-section be sure to enter the facet indices that correspond to the perimeter of your cross-section.

3.4 Merging Geometries

If you wish to merge multiple *Geometry* objects into a single object, you can use the *MergedSection* class:

class sectionproperties.pre.sections.**MergedSection**(*sections*)

Bases: *sectionproperties.pre.sections.Geometry*

Merges a number of section geometries into one geometry. Note that for the meshing algorithm to work, there needs to be connectivity between all regions of the provided geometries. Overlapping of geometries is permitted.

Parameters **sections** (list[*Geometry*]) – A list of geometry objects to merge into one *Geometry* object

The following example creates a combined cross-section with a 150x100x6 RHS placed on its side on top of a 200UB25.4. A mesh is generated with a maximum triangle size of 5.0 for the I-section and 2.5 for the RHS:

```
import sectionproperties.pre.sections as sections

isection = sections.ISection(d=203, b=133, t_f=7.8, t_w=5.8, r=8.9, n_r=8)
box = sections.Rhs(d=100, b=150, t=6, r_out=15, n_r=8, shift=[-8.5, 203])

geometry = sections.MergedSection([isection, box])
geometry.clean_geometry()
mesh = geometry.create_mesh(mesh_sizes=[5.0, 2.5])
```

Note: There must be connectivity between the *Geometry* objects that you wish to merge. It is currently not possible to analyse a cross-section that is composed of two or more unconnected domains.

Note: You may need to overwrite the perimeter facets list if predefined sections are used. Enabling labels while plotting the geometry is an easy way to manually identify the facet indices that make up the perimeter of the cross-section.

3.5 Cleaning the Geometry

When creating a merged section often there are overlapping facets or duplicate nodes. These geometry artefacts can cause difficulty for the meshing algorithm. It is therefore recommended to clean the geometry after merging sections which may result in overlapping or intersecting facets, or duplicate nodes. Cleaning the geometry can be carried out by using the *clean_geometry()* method:

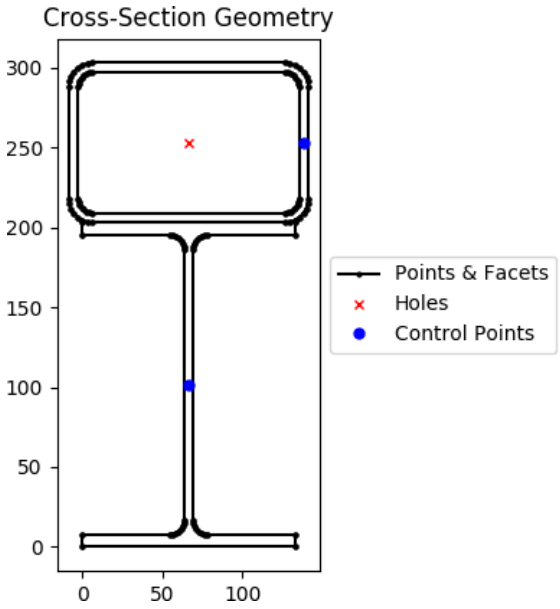
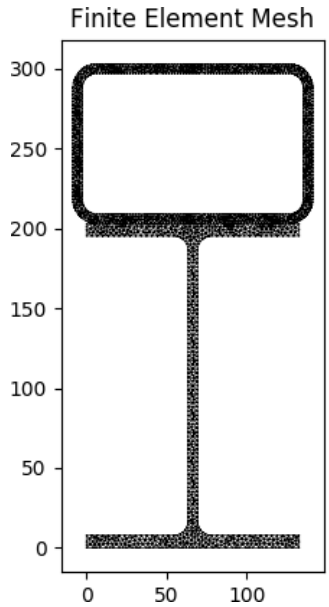


Fig. 35: Merged section geometry.



Geometry.**clean_geometry**(*verbose=False*)

Peforms a full clean on the geometry.

Parameters **verbose** (*bool*) – If set to true, information related to the geometry cleaning process is printed to the terminal.

Note: Cleaning the geometry is always recommended when creating a merged section, which may result in overlapping or intersecting facets, or duplicate nodes.

3.6 Perimeter Offset

The perimeter of a cross-section geometry can be offset by using the *offset_perimeter()* method:

sectionproperties.pre.offset.**offset_perimeter**(*geometry*, *offset*, *side='left'*, *plot_offset=False*)

Offsets the perimeter of a geometry of a *Geometry* object by a certain distance. Note that the perimeter facet list must be entered in a consecutive order.

Parameters

- **geometry** (*Geometry*) – Cross-section geometry object
- **offset** (*float*) – Offset distance for the perimeter
- **side** (*string*) – Side of the perimeter offset, either 'left' or 'right'. E.g. 'left' for a counter-clockwise offsets the perimeter inwards.
- **plot_offset** (*bool*) – If set to True, generates a plot comparing the old and new geometry

The following example 'corrodes' a 200UB25 I-section by 1.5 mm and compares a few of the section properties:

```
import sectionproperties.pre.sections as sections
from sectionproperties.pre.offset import offset_perimeter
from sectionproperties.analysis.cross_section import CrossSection

# calculate original section properties
original_geometry = sections.ISection(d=203, b=133, t_f=7.8, t_w=5.8, r=8.9, n_
↪r=16)
original_mesh = original_geometry.create_mesh(mesh_sizes=[3.0])
original_section = CrossSection(original_geometry, original_mesh)
original_section.calculate_geometric_properties()
original_area = original_section.get_area()
(original_ixx, _, _) = original_section.get_ic()

# calculate corroded section properties
corroded_geometry = offset_perimeter(original_geometry, 1.5, plot_offset=True)
corroded_mesh = corroded_geometry.create_mesh(mesh_sizes=[3.0])
corroded_section = CrossSection(corroded_geometry, corroded_mesh)
corroded_section.calculate_geometric_properties()
corroded_area = corroded_section.get_area()
(corroded_ixx, _, _) = corroded_section.get_ic()

# compare section properties
print("Area reduction = {0:.2f}%".format(
    100 * (original_area - corroded_area) / original_area))
```

(continues on next page)

(continued from previous page)

```
print("Ixx reduction = {0:.2f}%".format(
    100 *(original_ixx - corroded_ixx) / original_ixx))
```

The following plot is generated by the above example:

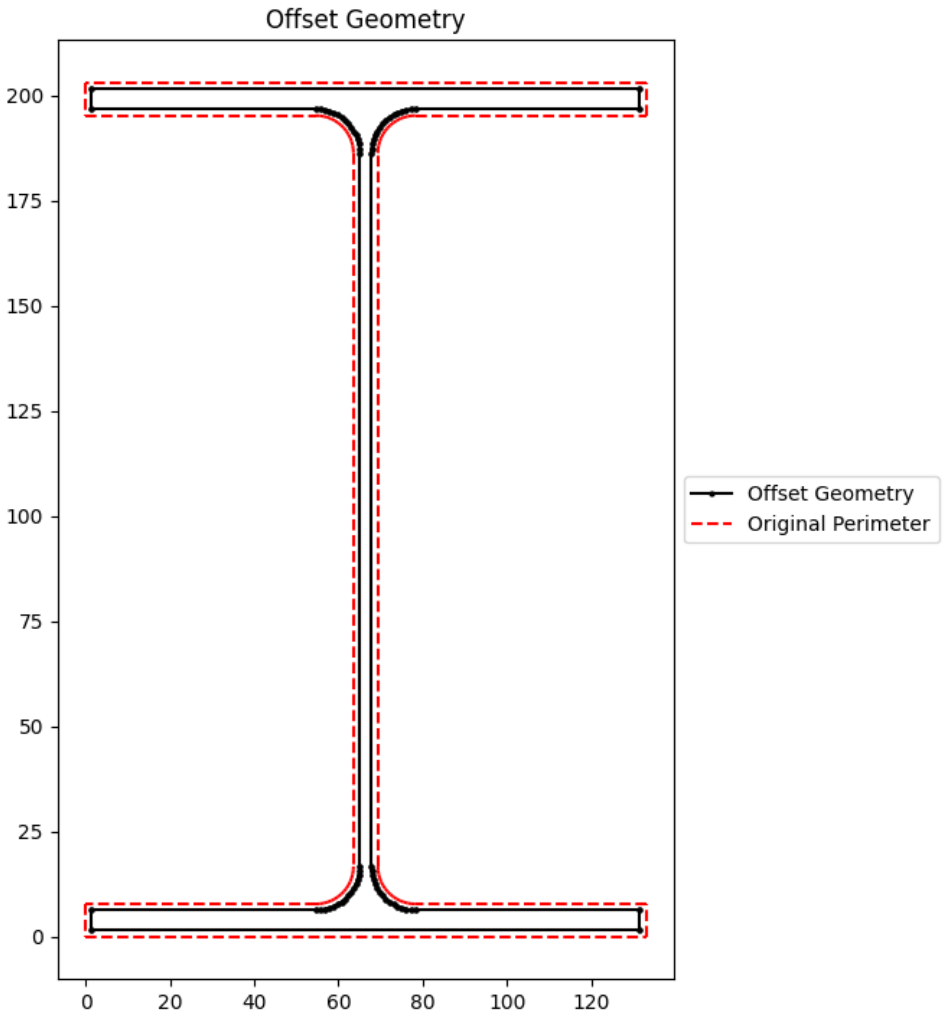


Fig. 36: 200UB25 with 1.5 mm corrosion.

The following is printed to the terminal:

```
Area reduction = 41.97%
Ixx reduction = 39.20%
```

Note: All the built-in sections in the `sections` module are built using an anti-clockwise facet direction. As a result, side='left' will reduce the cross-section, while side='right' will increase the cross-section.

Note: The `control_points` may need to be manually re-assigned if reducing the cross-section moves the control_point outside the geometry.

Warning: This feature is a *beta* addition and as a result may produce some errors if the offsetting drastically changes the geometry.

3.7 Visualising the Geometry

Geometry objects can be visualised by using the *plot_geometry()* method:

Geometry.**plot_geometry**(*ax=None*, *pause=True*, *labels=False*, *perimeter=False*)

Plots the geometry defined by the input section. If no axes object is supplied a new figure and axis is created.

Parameters

- **ax** (matplotlib.axes.Axes) – Axes object on which the mesh is plotted
- **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.
- **labels** (*bool*) – If set to true, node and facet labels are displayed
- **perimeter** (*bool*) – If set to true, boldens the perimeter of the cross-section

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example creates a CHS discretised with 64 points, with a diameter of 48 and thickness of 3.2, and plots the geometry:

```
import sectionproperties.pre.sections as sections

geometry = sections.Chs(d=48, t=3.2, n=64)
geometry.plot_geometry()
```

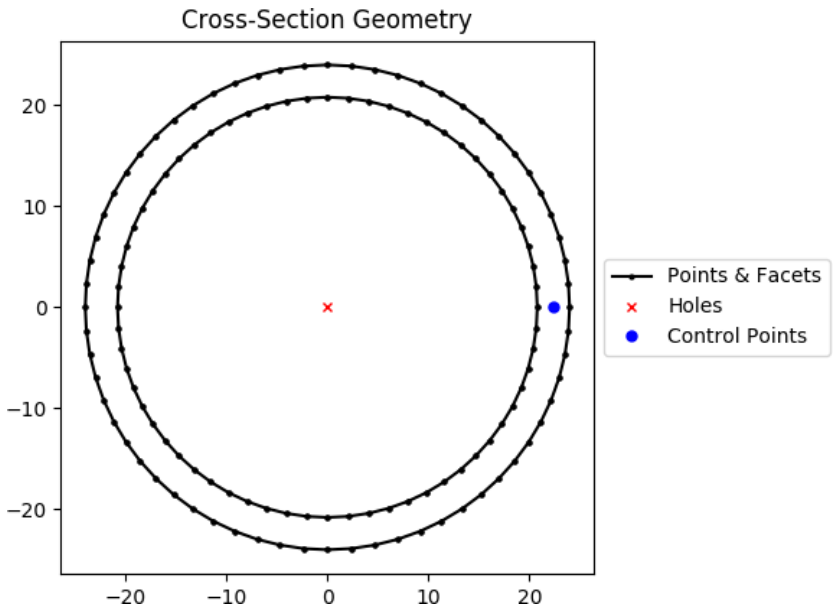


Fig. 37: Geometry generated by the above example.

3.8 Generating a Mesh

A finite element mesh is required to perform a cross-section analysis. A finite element mesh can be created by using the *create_mesh()* method:

Geometry.**create_mesh**(*mesh_sizes*)

Creates a quadratic triangular mesh from the Geometry object.

Parameters **mesh_sizes** – A list of maximum element areas corresponding to each region within the cross-section geometry.

Returns Object containing generated mesh data

Return type meshpy.triangle.MeshInfo

Raises **AssertionError** – If the number of mesh sizes does not match the number of regions

The following example creates a circular cross-section with a diameter of 50 with 64 points, and generates a mesh with a maximum triangular area of 2.5:

```
import sectionproperties.pre.sections as sections

geometry = sections.CircularSection(d=50, n=64)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
```

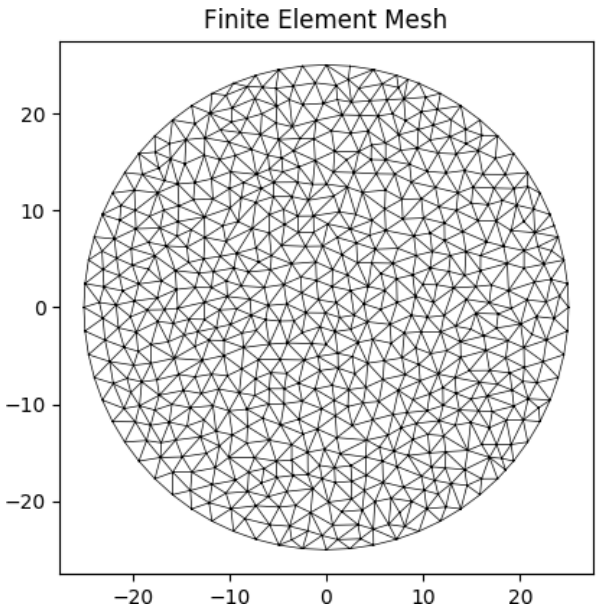


Fig. 38: Mesh generated from the above geometry.

Warning: The length of mesh_sizes must match the number of regions in the geometry object.

3.9 Defining Material Properties

Composite cross-sections can be analysed by specifying different material properties for each section of the mesh. Materials are defined in *sectionproperties* by creating a *Material* object:

class sectionproperties.pre.pre.**Material**(*name*, *elastic_modulus*, *poissons_ratio*, *yield_strength*, *color='w'*)

Bases: object

Class for structural materials.

Provides a way of storing material properties related to a specific material. The color can be a multitude of different formats, refer to https://matplotlib.org/api/colors_api.html and https://matplotlib.org/examples/color/named_colors.html for more information.

Parameters

- **name** (*string*) – Material name
- **elastic_modulus** (*float*) – Material modulus of elasticity
- **poissons_ratio** (*float*) – Material Poisson's ratio
- **yield_strength** (*float*) – Material yield strength
- **color** (matplotlib.colors) – Material color for rendering

Variables

- **name** (*string*) – Material name
- **elastic_modulus** (*float*) – Material modulus of elasticity
- **poissons_ratio** (*float*) – Material Poisson's ratio
- **shear_modulus** (*float*) – Material shear modulus, derived from the elastic modulus and Poisson's ratio assuming an isotropic material
- **yield_strength** (*float*) – Material yield strength
- **color** (matplotlib.colors) – Material color for rendering

The following example creates materials for concrete, steel and timber:

```
from sectionproperties.pre.pre import Material

concrete = Material(
    name='Concrete', elastic_modulus=30.1e3, poissons_ratio=0.2, yield_
↪strength=32,
        color='lightgrey'
)
steel = Material(
    name='Steel', elastic_modulus=200e3, poissons_ratio=0.3, yield_strength=500,
        color='grey'
)
timber = Material(
    name='Timber', elastic_modulus=8e3, poissons_ratio=0.35, yield_strength=20,
        color='burlywood'
)
```

CHAPTER 4

Running an Analysis

The first step in running a cross-section analysis is the creation of a *CrossSection* object. This class stores the structural geometry and finite element mesh and provides methods to perform various types of cross-section analyses.

class sectionproperties.analysis.cross_section.**CrossSection**(*geometry*, *mesh*, *materials=None*, *time_info=False*)

Bases: object

Class for structural cross-sections.

Stores the finite element geometry, mesh and material information and provides methods to compute the cross-section properties. The element type used in this program is the six-noded quadratic triangular element.

The constructor extracts information from the provided mesh object and creates and stores the corresponding Tri6 finite element objects.

Parameters

- **geometry** (*Geometry*) – Cross-section geometry object used to generate the mesh
- **mesh** (meshpy.triangle.MeshInfo) – Mesh object returned by meshpy
- **materials** (list[*Material*]) – A list of material properties corresponding to various regions in the geometry and mesh. Note that if materials are specified, the number of material objects ust equal the number of regions in the geometry. If no materials are specified, only a purely geometric analysis can take place, and all regions will be assigned a default material with an elastic modulus and yield strength equal to 1, and a Poisson's ratio equal to 0.
- **time_info** (*bool*) – If set to True, a detailed description of the computation and the time cost is printed to the terminal.

The following example creates a *CrossSection* object of a 100D x 50W rectangle using a mesh size of 5:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.RectangularSection(d=100, b=50)
```

(continues on next page)

(continued from previous page)

```
mesh = geometry.create_mesh(mesh_sizes=[5])
section = CrossSection(geometry, mesh)
```

The following example creates a 100D x 50W rectangle, with the top half of the section comprised of timber and the bottom half steel. The timber section is meshed with a maximum area of 10 and the steel section mesh with a maximum area of 5:

```
import sectionproperties.pre.sections as sections
from sectionproperties.pre.pre import Material
from sectionproperties.analysis.cross_section import CrossSection

geom_steel = sections.RectangularSection(d=50, b=50)
geom_timber = sections.RectangularSection(d=50, b=50, shift=[0, 50])
geometry = sections.MergedSection([geom_steel, geom_timber])
geometry.clean_geometry()

mesh = geometry.create_mesh(mesh_sizes=[5, 10])

steel = Material(
    name='Steel', elastic_modulus=200e3, poissons_ratio=0.3, yield_strength=250,
    color='grey'
)
timber = Material(
    name='Timber', elastic_modulus=8e3, poissons_ratio=0.35, yield_strength=20,
    color='burlywood'
)

section = CrossSection(geometry, mesh, [steel, timber])
section.plot_mesh(materials=True, alpha=0.5)
```

Variables

- **elements** (list[*Tri6*]) – List of finite element objects describing the cross-section mesh
- **num_nodes** (*int*) – Number of nodes in the finite element mesh
- **geometry** (*Geometry*) – Cross-section geometry object used to generate the mesh
- **mesh** (meshpy.triangle.MeshInfo) – Mesh object returned by meshpy
- **mesh_nodes** (numpy.ndarray) – Array of node coordinates from the mesh
- **mesh_elements** (numpy.ndarray) – Array of connectivities from the mesh
- **mesh_attributes** (numpy.ndarray) – Array of attributes from the mesh
- **materials** – List of materials
- **material_groups** – List of objects containing the elements in each defined material
- **section_props** (*SectionProperties*) – Class to store calculated section properties

Raises **AssertionError** – If the number of materials does not equal the number of regions

4.1 Checking the Mesh Quality

Before carrying out a cross-section analysis it is a good idea to check the quality of the finite element mesh. Some useful methods are provided to display mesh statistics and to plot the finite element mesh:

CrossSection.**display_mesh_info**()

Prints mesh statistics (number of nodes, elements and regions) to the command window.

The following example displays the mesh statistics for a Tee section merged from two rectangles:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

rec1 = sections.RectangularSection(d=100, b=25, shift=[-12.5, 0])
rec2 = sections.RectangularSection(d=25, b=100, shift=[-50, 100])
geometry = sections.MergedSection([rec1, rec2])
mesh = geometry.create_mesh(mesh_sizes=[5, 2.5])
section = CrossSection(geometry, mesh)
section.display_mesh_info()

>>>Mesh Statistics:
>>>--4920 nodes
>>>--2365 elements
>>>--2 regions
```

CrossSection.**plot_mesh**(*ax=None*, *pause=True*, *alpha=1*, *materials=False*, *mask=None*)

Plots the finite element mesh. If no axes object is supplied a new figure and axis is created.

Parameters

- **ax** (matplotlib.axes.Axes) – Axes object on which the mesh is plotted
- **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.
- **alpha** (*float*) – Transparency of the mesh outlines: $0 \leq \alpha \leq 1$
- **materials** (*bool*) – If set to true and material properties have been provided to the *CrossSection* object, shades the elements with the specified material colours
- **mask** (*list[bool]*) – Mask array, of length num_nodes, to mask out triangles

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the mesh generated for the second example listed under the *CrossSection* object definition:

```
import sectionproperties.pre.sections as sections
from sectionproperties.pre.pre import Material
from sectionproperties.analysis.cross_section import CrossSection

geom_steel = sections.RectangularSection(d=50, b=50)
geom_timber = sections.RectangularSection(d=50, b=50, shift=[50, 0])
geometry = sections.MergedSection([geom_steel, geom_timber])
geometry.clean_geometry()

mesh = geometry.create_mesh(mesh_sizes=[5, 10])

steel = Material(
    name='Steel', elastic_modulus=200e3, poissons_ratio=0.3, yield_strength=250,
    color='grey'
)
timber = Material(
    name='Timber', elastic_modulus=8e3, poissons_ratio=0.35, yield_strength=20,
```

(continues on next page)

(continued from previous page)

```
    color='burlywood'
)

section = CrossSection(geometry, mesh, [steel, timber])
section.plot_mesh(materials=True, alpha=0.5)
```

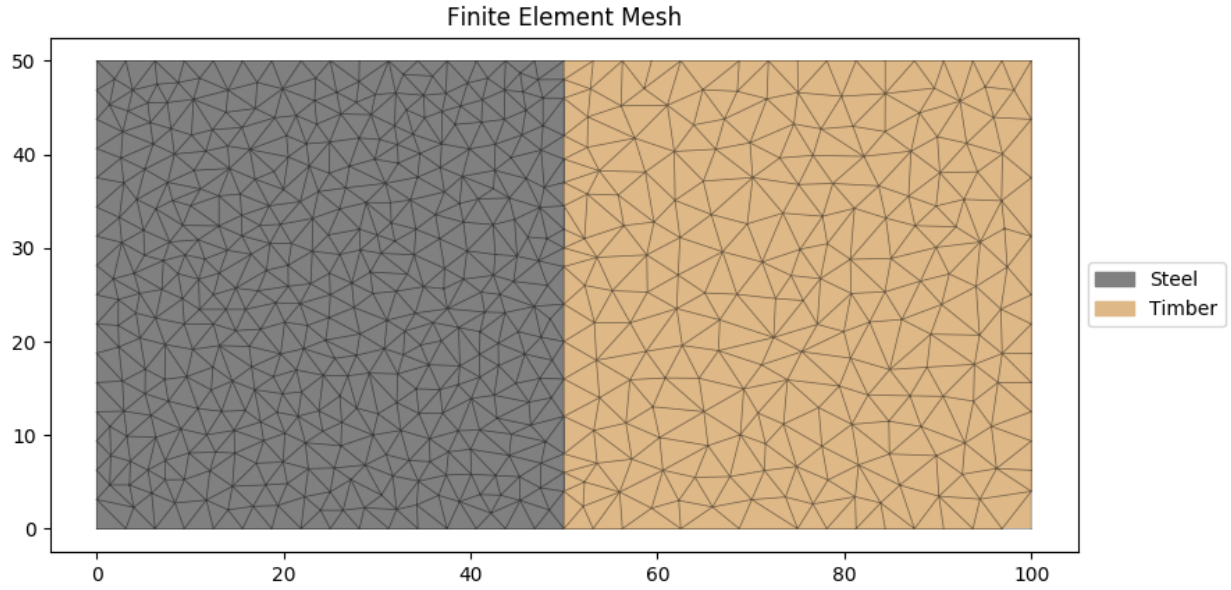


Fig. 1: Finite element mesh generated by the above example.

4.2 Geometric Analysis

A geometric analysis calculates the area properties of the cross-section.

`CrossSection.`**`calculate_geometric_properties`**(*time_info=False*)

Calculates the geometric properties of the cross-section and stores them in the *`SectionProperties`* object contained in the `section_props` class variable.

Parameters **`time_info`** (*`bool`*) – If set to True, a detailed description of the computation and the time cost is printed to the terminal.

The following geometric section properties are calculated:

- Cross-sectional area
- Cross-sectional perimeter
- Modulus weighted area (axial rigidity)
- First moments of area
- Second moments of area about the global axis
- Second moments of area about the centroidal axis
- Elastic centroid
- Centroidal section moduli

- Radii of gyration
- Principal axis properties

If materials are specified for the cross-section, the moments of area and section moduli are elastic modulus weighted.

The following example demonstrates the use of this method:

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
```

4.3 Plastic Analysis

A plastic analysis calculates the plastic properties of the cross-section.

Note: A geometric analysis must be performed on the CrossSection object before a plastic analysis is carried out.

CrossSection.**calculate_plastic_properties**(*time_info=False*, *verbose=False*, *debug=False*)

Calculates the plastic properties of the cross-section and stores the, in the *SectionProperties* object contained in the `section_props` class variable.

Parameters

- **time_info** (*bool*) – If set to True, a detailed description of the computation and the time cost is printed to the terminal.
- **verbose** (*bool*) – If set to True, the number of iterations required for each plastic axis is printed to the terminal.
- **debug** (*bool*) – If set to True, the geometry is plotted each time a new mesh is generated by the plastic centroid algorithm.

The following warping section properties are calculated:

- Plastic centroid for bending about the centroidal and principal axes
- Plastic section moduli for bending about the centroidal and principal axes
- Shape factors for bending about the centroidal and principal axes

If materials are specified for the cross-section, the plastic section moduli are displayed as plastic moments (i.e $M_p = f_y S$) and the shape factors are not calculated.

Note that the geometric properties must be calculated before the plastic properties are calculated:

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
```

Raises **RuntimeError** – If the geometric properties have not been calculated prior to calling this method

4.4 Warping Analysis

A warping analysis calculates the torsion and shear properties of the cross-section.

Note: A geometric analysis must be performed on the CrossSection object before a warping analysis is carried out.

CrossSection.**calculate_warping_properties**(*time_info=False*, *solver_type='direct'*)

Calculates all the warping properties of the cross-section and stores them in the *SectionProperties* object contained in the section_props class variable.

Parameters

- **time_info** (*bool*) – If set to True, a detailed description of the computation and the time cost is printed to the terminal.
- **solver_type** (*string*) – Solver used for solving systems of linear equations, either using the *'direct'* method or *'cgs'* iterative method

The following warping section properties are calculated:

- Torsion constant
- Shear centre
- Shear area
- Warping constant
- Monosymmetry constant

If materials are specified, the values calculated for the torsion constant, warping constant and shear area are elastic modulus weighted.

Note that the geometric properties must be calculated first for the calculation of the warping properties to be correct:

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
```

Raises **RuntimeError** – If the geometric properties have not been calculated prior to calling this method

4.5 Stress Analysis

A stress analysis calculates the cross-section stresses arising from a set of forces and moments. Executing this method returns a *StressResult* object which stores the cross-section stresses and provides stress plotting functions.

Note: A geometric and warping analysis must be performed on the CrossSection object before a stress analysis is carried out.

CrossSection.**calculate_stress**(*N=0*, *Vx=0*, *Vy=0*, *Mxx=0*, *Myy=0*, *M11=0*, *M22=0*, *Mzz=0*, *time_info=False*)

Calculates the cross-section stress resulting from design actions and returns a *StressPost* object allowing post-processing of the stress results.

Parameters

- **N** (*float*) – Axial force
- **Vx** (*float*) – Shear force acting in the x-direction
- **Vy** (*float*) – Shear force acting in the y-direction
- **Mxx** (*float*) – Bending moment about the centroidal xx-axis
- **Myy** (*float*) – Bending moment about the centroidal yy-axis
- **M11** (*float*) – Bending moment about the centroidal 11-axis
- **M22** (*float*) – Bending moment about the centroidal 22-axis
- **Mzz** (*float*) – Torsion moment about the centroidal zz-axis
- **time_info** (*bool*) – If set to True, a detailed description of the computation and the time cost is printed to the terminal.

Returns Object for post-processing cross-section stresses

Return type *StressPost*

Note that a geometric and warping analysis must be performed before a stress analysis is carried out:

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(N=1e3, Vy=3e3, Mxx=1e6)
```

Raises **RuntimeError** – If a geometric and warping analysis have not been performed prior to calling this method

4.6 Calculating Frame Properties

Calculates the cross-section properties required for a 2D or 3D frame analysis.

Note: This method is significantly faster than performing a geometric and a warping analysis and has no prerequisites.

CrossSection.**calculate_frame_properties**(*time_info=False*, *solver_type='direct'*)

Calculates and returns the properties required for a frame analysis. The properties are also stored in the *SectionProperties* object contained in the section_props class variable.

Parameters

- **time_info** (*bool*) – If set to True, a detailed description of the computation and the time cost is printed to the terminal.
- **solver_type** (*string*) – Solver used for solving systems of linear equations, either using the *'direct'* method or *'cgs'* iterative method

Returns Cross-section properties to be used for a frame analysis *(area, ixx, iyy, ixy, j, phi)*

Return type tuple(float, float, float, float, float, float)

The following section properties are calculated:

- Cross-sectional area *(area)*

- Second moments of area about the centroidal axis *(ixx, iyy, ixy)*
- Torsion constant *(j)*
- Principal axis angle *(phi)*

If materials are specified for the cross-section, the area, second moments of area and torsion constant are elastic moulus weighted.

The following example demonstrates the use of this method:

```
section = CrossSection(geometry, mesh)
(area, ixx, iyy, ixy, j, phi) = section.calculate_frame_properties()
```

CHAPTER 5

Viewing the Results

5.1 Printing a List of the Section Properties

A list of section properties that have been calculated by various analyses can be printed to the terminal using the *display_results()* method that belongs to every *CrossSection* object.

CrossSection.**display_results**(*fmt='8.6e'*)

Prints the results that have been calculated to the terminal.

Parameters **fmt** (*string*) – Number formatting string

The following example displays the geometric section properties for a 100D x 50W rectangle with three digits after the decimal point:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.RectangularSection(d=100, b=50)
mesh = geometry.create_mesh(mesh_sizes=[5])

section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()

section.display_results(fmt='.3f')
```

5.2 Getting Specific Section Properties

Alternatively, there are a number of methods that can be called on the *CrossSection* object to return a specific section property:

5.2.1 Cross-Section Area

CrossSection.**get_area**()

Returns Cross-section area

Return type float

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
area = section.get_area()
```

5.2.2 Cross-Section Perimeter

CrossSection.**get_perimeter**()

Returns Cross-section perimeter

Return type float

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
perimeter = section.get_perimeter()
```

5.2.3 Axial Rigidity

If material properties have been specified, returns the axial rigidity of the section.

CrossSection.**get_ea**()

Returns Modulus weighted area (axial rigidity)

Return type float

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
ea = section.get_ea()
```

5.2.4 First Moments of Area

CrossSection.**get_q**()

Returns First moments of area about the global axis *(qx, qy)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(qx, qy) = section.get_q()
```

5.2.5 Second Moments of Area

CrossSection.**get_ig**()

Returns Second moments of area about the global axis *(ixx_g, iyy_g, ixy_g)*

Return type tuple(float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(ixx_g, iyy_g, ixy_g) = section.get_ig()
```

CrossSection.**get_ic**()

Returns Second moments of area centroidal axis *(ixx_c, iyy_c, ixy_c)*

Return type tuple(float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(ixx_c, iyy_c, ixy_c) = section.get_ic()
```

CrossSection.**get_ip**()

Returns Second moments of area about the principal axis *(i11_c, i22_c)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(i11_c, i22_c) = section.get_ip()
```

5.2.6 Elastic Centroid

CrossSection.**get_c**()

Returns Elastic centroid *(cx, cy)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(cx, cy) = section.get_c()
```

5.2.7 Section Moduli

CrossSection.**get_z**()

Returns Elastic section moduli about the centroidal axis with respect to the top and bottom fibres *(zxx_plus, zxx_minus, zyy_plus, zyy_minus)*

Return type tuple(float, float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(zxx_plus, zxx_minus, zyy_plus, zyy_minus) = section.get_z()
```

CrossSection.**get_zp**()

Returns Elastic section moduli about the principal axis with respect to the top and bottom fibres *(z11_plus, z11_minus, z22_plus, z22_minus)*

Return type tuple(float, float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(z11_plus, z11_minus, z22_plus, z22_minus) = section.get_zp()
```

5.2.8 Radii of Gyration

CrossSection.**get_rc**()

Returns Radii of gyration about the centroidal axis *(rx, ry)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(rx, ry) = section.get_rc()
```

CrossSection.**get_rp**()

Returns Radii of gyration about the principal axis *(r11, r22)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(r11, r22) = section.get_rp()
```

5.2.9 Principal Axis Angle

CrossSection.**get_phi**()

Returns Principal bending axis angle

Return type float

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
phi = section.get_phi()
```

5.2.10 Torsion Constant

CrossSection.**get_j**()

Returns St. Venant torsion constant

Return type float

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
j = section.get_j()
```

5.2.11 Shear Centre

CrossSection.**get_sc**()

Returns Centroidal axis shear centre (elasticity approach) *(x_se, y_se)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
(x_se, y_se) = section.get_sc()
```

CrossSection.**get_sc_p**()

Returns Principal axis shear centre (elasticity approach) *(x11_se, y22_se)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
(x11_se, y22_se) = section.get_sc_p()
```

5.2.12 Trefftz's Shear Centre

CrossSection.**get_sc_t**()

Returns Centroidal axis shear centre (Trefftz's approach) *(x_st, y_st)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
(x_st, y_st) = section.get_sc_t()
```

5.2.13 Warping Constant

CrossSection.**get_gamma**()

Returns Warping constant

Return type float

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
gamma = section.get_gamma()
```

5.2.14 Shear Area

CrossSection.**get_As**()

Returns Shear area for loading about the centroidal axis *(A_sx, A_sy)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
(A_sx, A_sy) = section.get_As()
```

CrossSection.**get_As_p**()

Returns Shear area for loading about the principal bending axis *(A_s11, A_s22)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
(A_s11, A_s22) = section.get_As_p()
```

5.2.15 Monosymmetry Constants

CrossSection.**get_beta**()

Returns Monosymmetry constant for bending about both global axes *(beta_x_plus, beta_x_minus, beta_y_plus, beta_y_minus)*. The *plus* value relates to the top flange in compression and the *minus* value relates to the bottom flange in compression.

Return type tuple(float, float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
(beta_x_plus, beta_x_minus, beta_y_plus, beta_y_minus) = section.get_beta()
```

CrossSection.**get_beta_p**()

Returns Monosymmetry constant for bending about both principal axes *(beta_11_plus, beta_11_minus, beta_22_plus, beta_22_minus)*. The *plus* value relates to the top flange in compression and the *minus* value relates to the bottom flange in compression.

Return type tuple(float, float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
(beta_11_plus, beta_11_minus, beta_22_plus, beta_22_minus) = section.get_beta_p()
```

5.2.16 Plastic Centroid

CrossSection.**get_pc**()

Returns Centroidal axis plastic centroid *(x_pc, y_pc)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
(x_pc, y_pc) = section.get_pc()
```

CrossSection.**get_pc_p**()

Returns Principal bending axis plastic centroid *(x11_pc, y22_pc)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
(x11_pc, y22_pc) = section.get_pc_p()
```

5.2.17 Plastic Section Moduli

CrossSection.**get_s**()

Returns Plastic section moduli about the centroidal axis *(sxx, syy)*

Return type tuple(float, float)

If material properties have been specified, returns the plastic moment $M_p = f_y S$.

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
(sxx, syy) = section.get_s()
```

CrossSection.**get_sp**()

Returns Plastic section moduli about the principal bending axis *(s11, s22)*

Return type tuple(float, float)

If material properties have been specified, returns the plastic moment $M_p = f_y S$.

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
(s11, s22) = section.get_sp()
```

5.2.18 Shape Factors

CrossSection.**get_sf**()

Returns Centroidal axis shape factors with respect to the top and bottom fibres *(sf_xx_plus, sf_xx_minus, sf_yy_plus, sf_yy_minus)*

Return type tuple(float, float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
(sf_xx_plus, sf_xx_minus, sf_yy_plus, sf_yy_minus) = section.get_sf()
```

CrossSection.**get_sf_p**()

Returns Principal bending axis shape factors with respect to the top and bottom fibres *(sf_11_plus, sf_11_minus, sf_22_plus, sf_22_minus)*

Return type tuple(float, float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
(sf_11_plus, sf_11_minus, sf_22_plus, sf_22_minus) = section.get_sf_p()
```

5.3 Section Property Centroids Plots

A plot of the centroids (elastic, plastic and shear centre) can be produced with the finite element mesh in the background:

CrossSection.**plot_centroids**(*pause=True*)

Plots the elastic centroid, the shear centre, the plastic centroids and the principal axis, if they have been calculated, on top of the finite element mesh.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example analyses a 200 PFC section and displays a plot of the centroids:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.PfcSection(d=200, b=75, t_f=12, t_w=6, r=12, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])

section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
section.calculate_plastic_properties()

section.plot_centroids()
```

The following example analyses a 150x90x12 UA section and displays a plot of the centroids:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])

section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
section.calculate_plastic_properties()

section.plot_centroids()
```

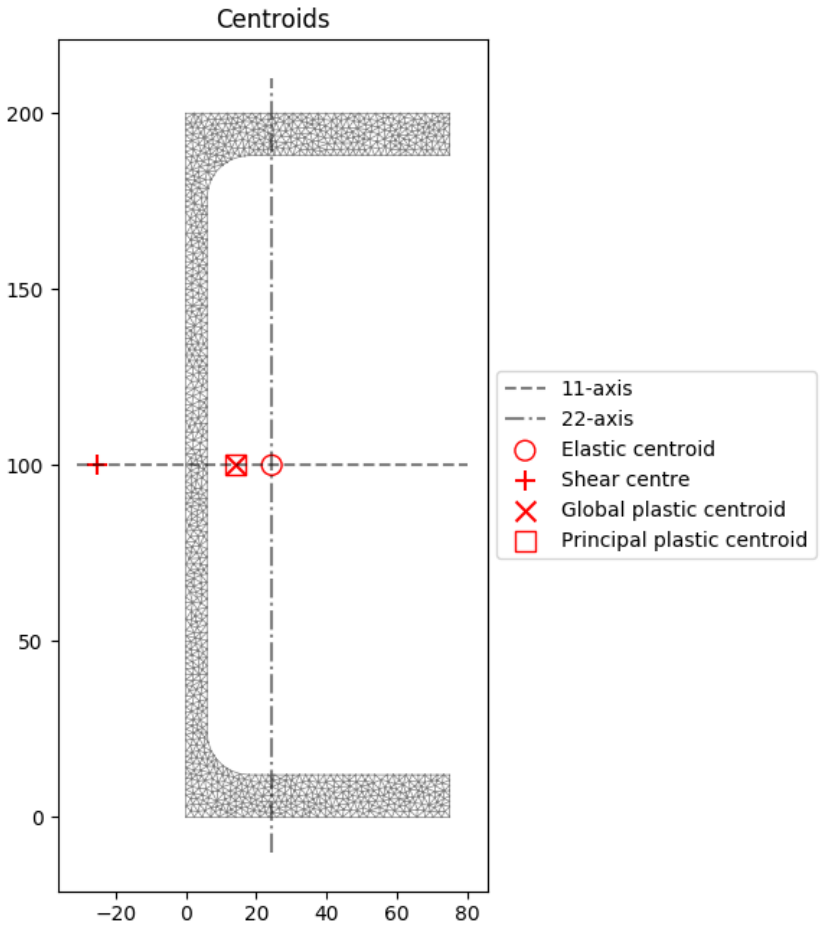


Fig. 1: Plot of the centroids generated by the above example.

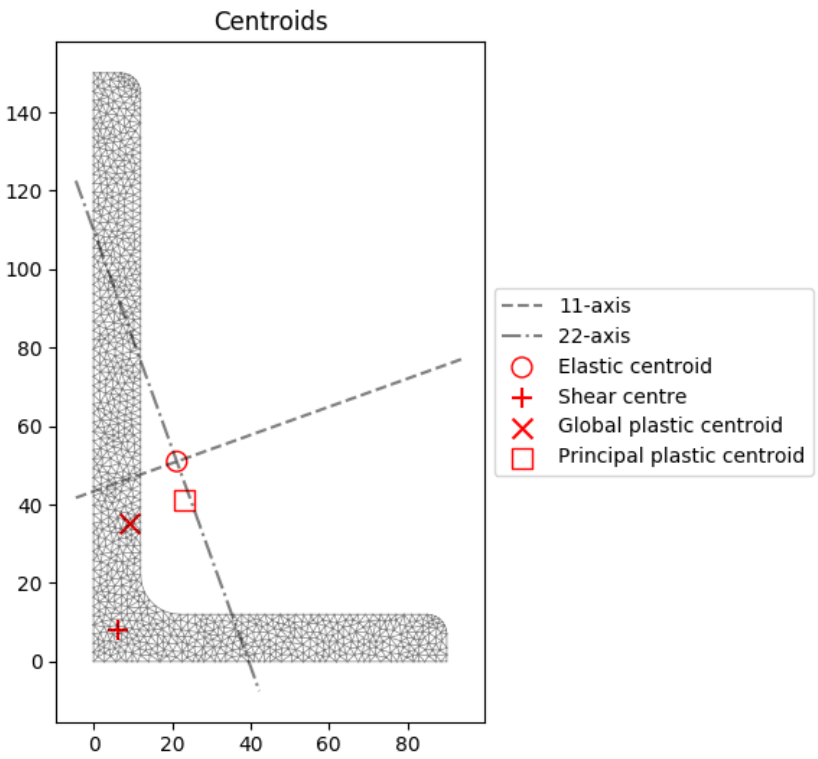


Fig. 2: Plot of the centroids generated by the above example.

5.4 Plotting Cross-Section Stresses

There are a number of methods that can be called from a `StressResult` object to plot the various cross-section stresses. These methods take the following form:

`StressResult`.plot_(*stress/vector*)_(*action*)_(*stresstype*)

where:

- *stress* denotes a contour plot and *vector* denotes a vector plot.
- *action* denotes the type of action causing the stress e.g. *mxx* for bending moment about the x-axis. Note that the action is omitted for stresses caused by the application of all actions.
- *stresstype* denotes the type of stress that is being plotted e.g. *zx* for the *x*-component of shear stress.

The examples shown in the methods below are performed on a 150x90x12 UA (unequal angle) section. The `CrossSection` object is created below:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)
```

5.4.1 Primary Stress Plots

Axial Stress ($\sigma_{zz,N}$)

StressPost.**plot_stress_n_zz**(*pause=True*)

Produces a contour plot of the normal stress $\sigma_{zz,N}$ resulting from the axial load N.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the normal stress within a 150x90x12 UA section resulting from an axial force of 10 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(N=10e3)

stress_post.plot_stress_n_zz()
```

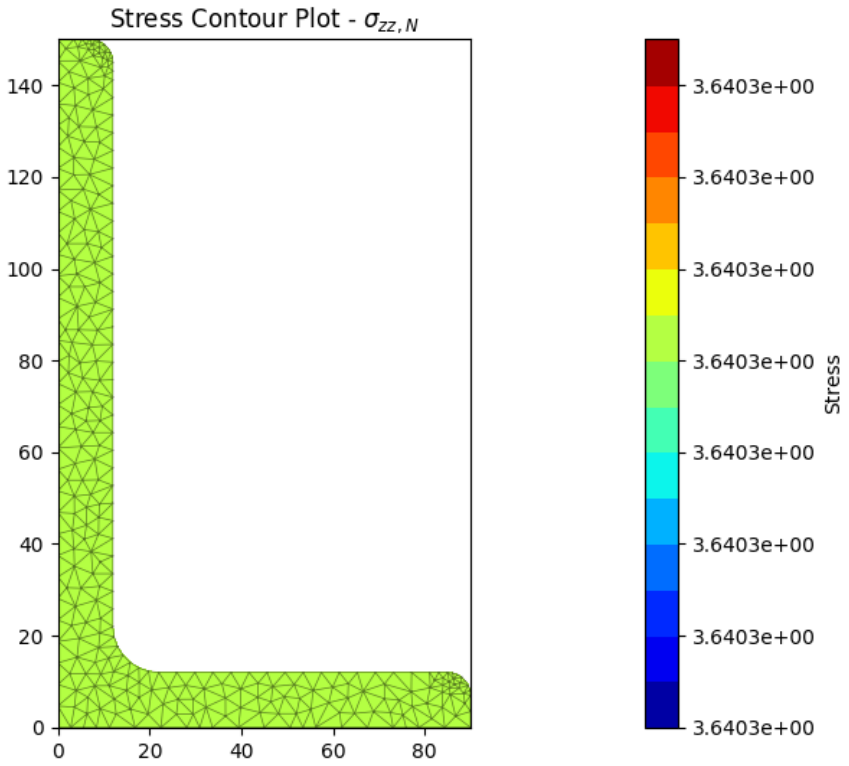


Fig. 3: Contour plot of the axial stress.

Bending Stress ($\sigma_{zz,Mxx}$)

StressPost.**plot_stress_mxx_zz**(*pause=True*)

Produces a contour plot of the normal stress $\sigma_{zz,Mxx}$ resulting from the bending moment M_{xx}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the normal stress within a 150x90x12 UA section resulting from a bending moment about the x-axis of 5 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mxx=5e6)

stress_post.plot_stress_mxx_zz()
```

Bending Stress ($\sigma_{zz,Myy}$)

StressPost.**plot_stress_myy_zz**(*pause=True*)

Produces a contour plot of the normal stress $\sigma_{zz,Myy}$ resulting from the bending moment M_{yy}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the normal stress within a 150x90x12 UA section resulting from a bending moment about the y-axis of 2 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Myy=2e6)

stress_post.plot_stress_myy_zz()
```

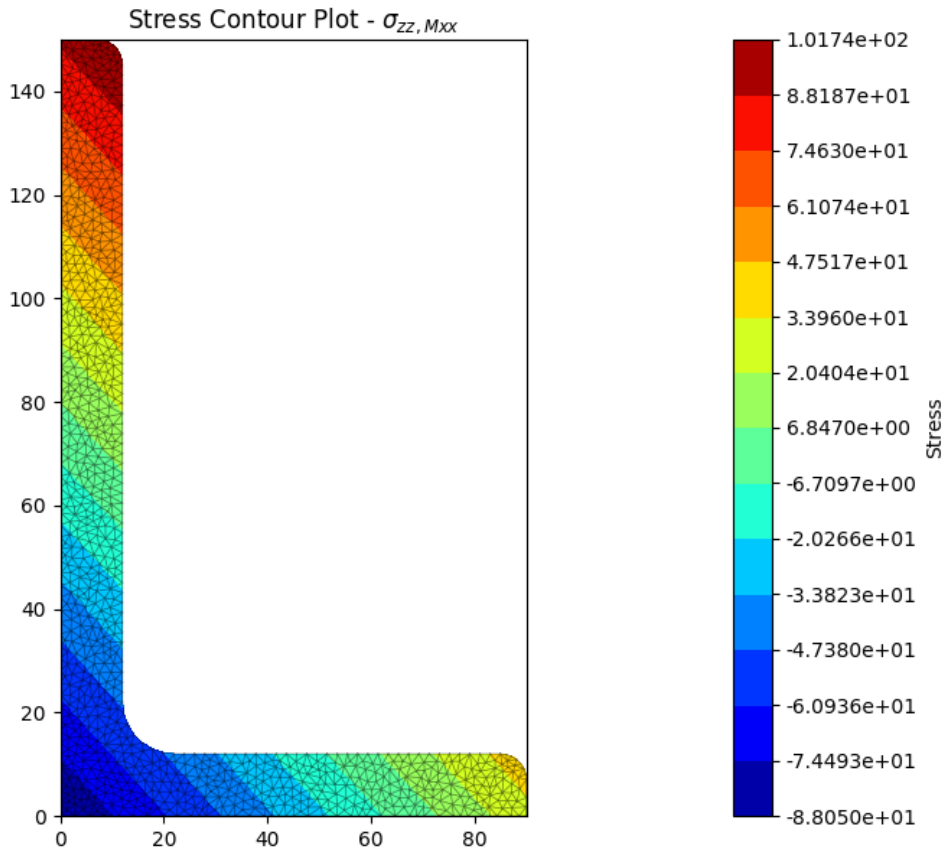


Fig. 4: Contour plot of the bending stress.

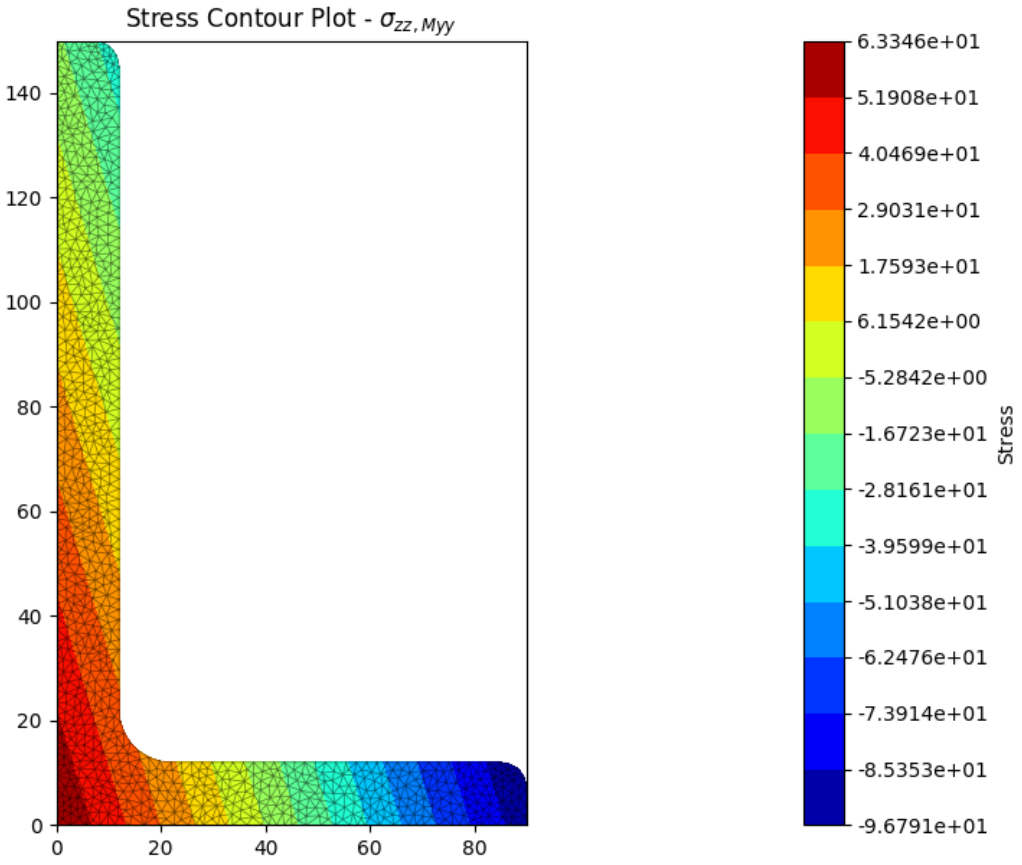


Fig. 5: Contour plot of the bending stress.

Bending Stress ($\sigma_{zz,M11}$)

StressPost.**plot_stress_m11_zz**(*pause=True*)

Produces a contour plot of the normal stress $\sigma_{zz,M11}$ resulting from the bending moment M_{11}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the normal stress within a 150x90x12 UA section resulting from a bending moment about the 11-axis of 5 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(M11=5e6)

stress_post.plot_stress_m11_zz()
```

Bending Stress ($\sigma_{zz,M22}$)

StressPost.**plot_stress_m22_zz**(*pause=True*)

Produces a contour plot of the normal stress $\sigma_{zz,M22}$ resulting from the bending moment M_{22}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the normal stress within a 150x90x12 UA section resulting from a bending moment about the 22-axis of 2 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(M22=5e6)

stress_post.plot_stress_m22_zz()
```

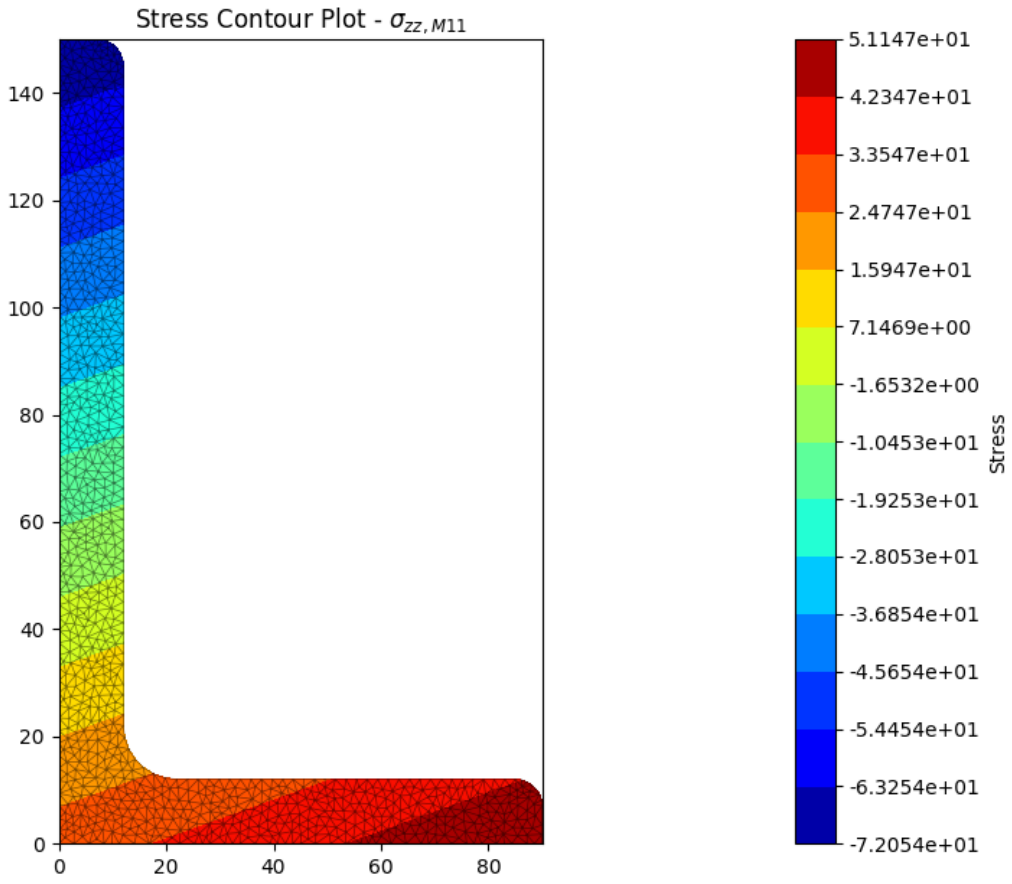


Fig. 6: Contour plot of the bending stress.

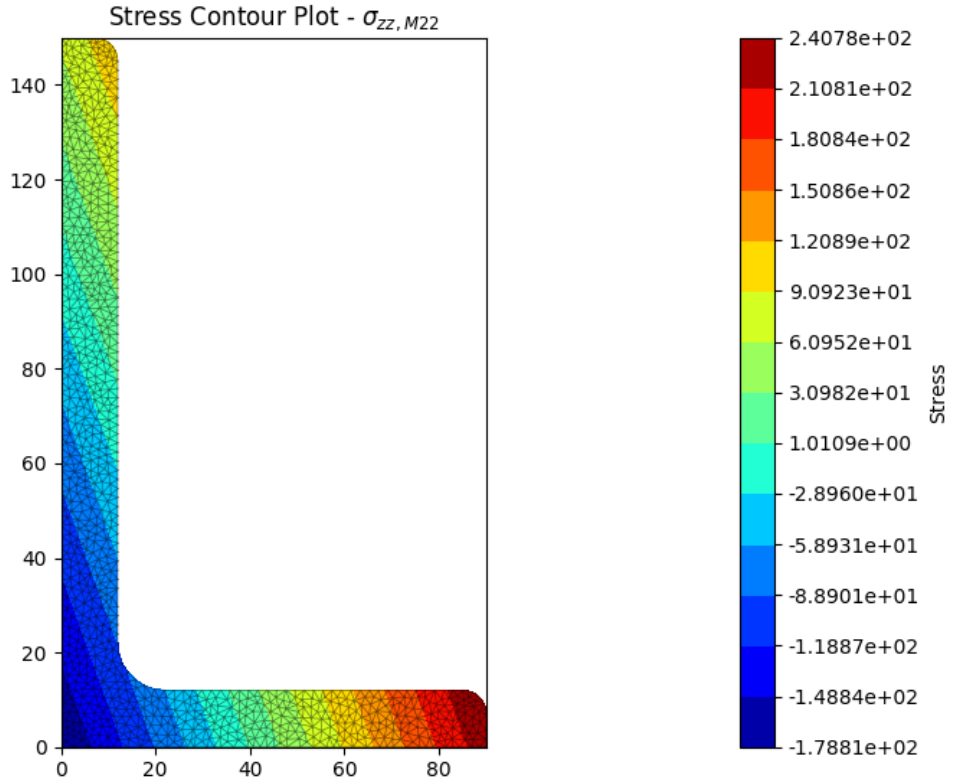


Fig. 7: Contour plot of the bending stress.

Bending Stress ($\sigma_{zz,\Sigma M}$)

StressPost.**plot_stress_m_zz**(*pause=True*)

Produces a contour plot of the normal stress $\sigma_{zz,\Sigma M}$ resulting from all bending moments $M_{xx} + M_{yy} + M_{11} + M_{22}$.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the normal stress within a 150x90x12 UA section resulting from a bending moment about the x-axis of 5 kN.m, a bending moment about the y-axis of 2 kN.m and a bending moment of 3 kN.m about the 11-axis:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mxx=5e6, Myy=2e6, M11=3e6)

stress_post.plot_stress_m_zz()
```

Torsion Stress ($\sigma_{zx,Mzz}$)

StressPost.**plot_stress_mzz_zx**(*pause=True*)

Produces a contour plot of the *x*-component of the shear stress $\sigma_{zx,Mzz}$ resulting from the torsion moment M_{zz}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the x-component of the shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6)

stress_post.plot_stress_mzz_zx()
```

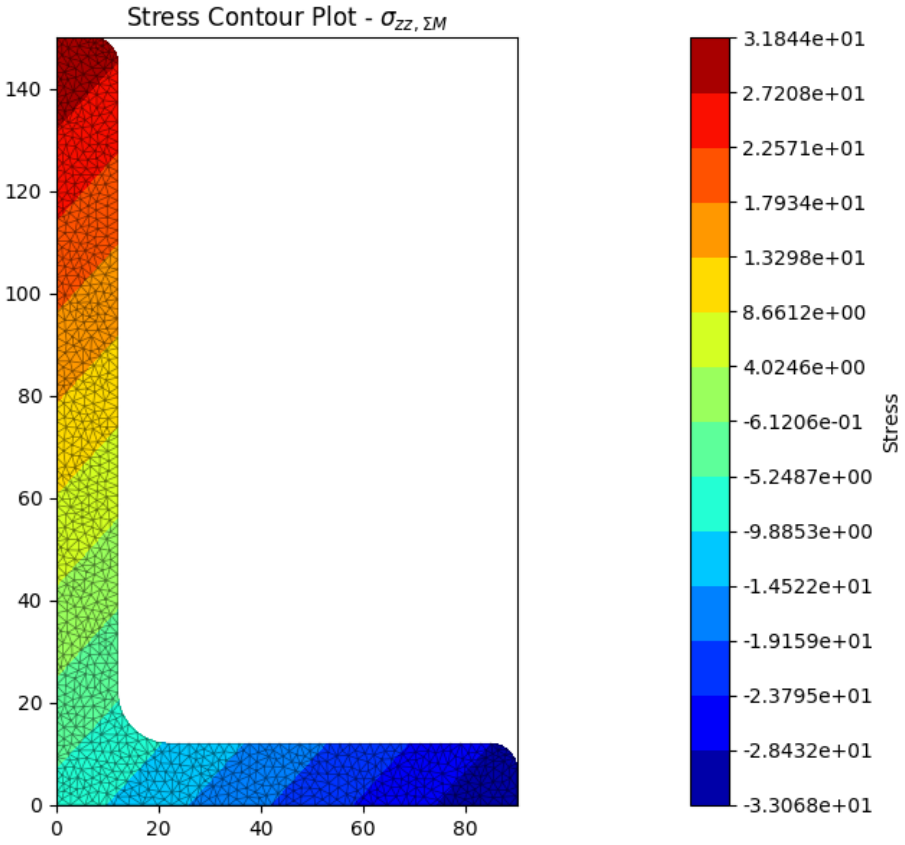


Fig. 8: Contour plot of the bending stress.

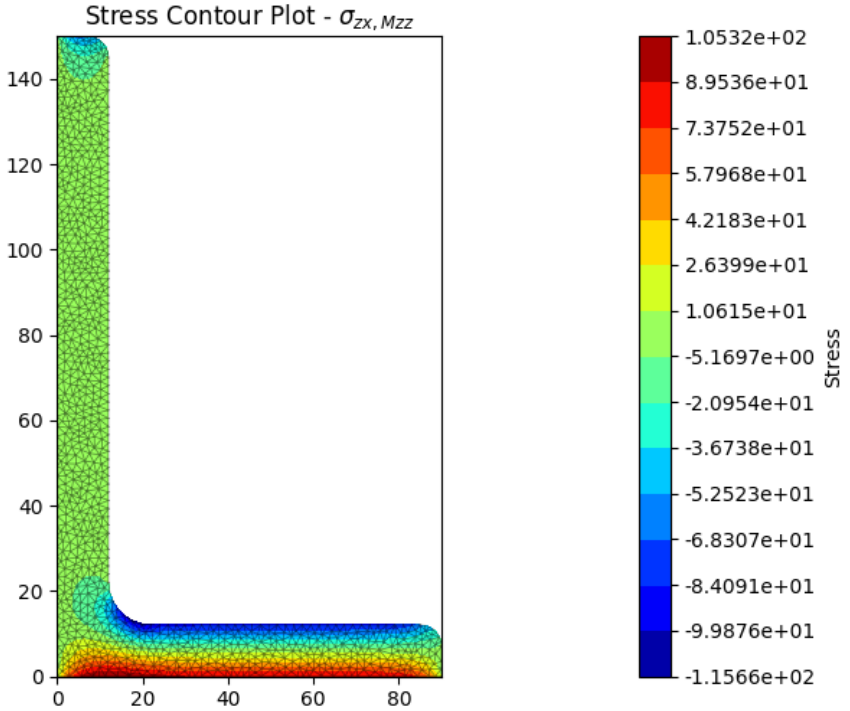


Fig. 9: Contour plot of the shear stress.

Torsion Stress ($\sigma_{zy,Mzz}$)

StressPost.**plot_stress_mzz_zy**(*pause=True*)

Produces a contour plot of the y-component of the shear stress $\sigma_{zy,Mzz}$ resulting from the torsion moment M_{zz}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the y-component of the shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6)

stress_post.plot_stress_mzz_zy()
```

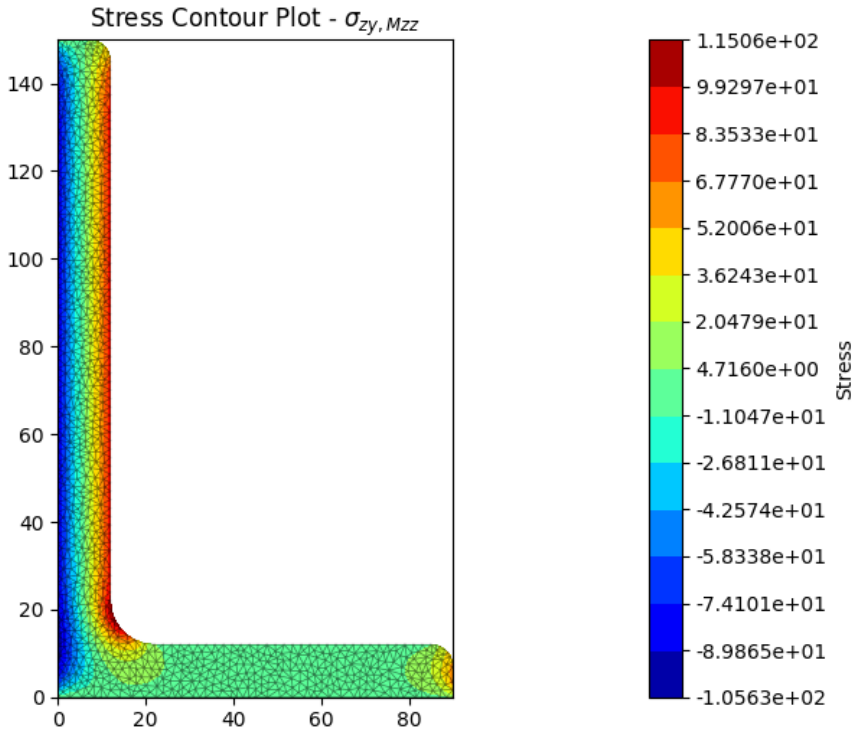


Fig. 10: Contour plot of the shear stress.

Torsion Stress ($\sigma_{zxy,Mzz}$)

StressPost.**plot_stress_mzz_zxy**(*pause=True*)

Produces a contour plot of the resultant shear stress $\sigma_{zxy,Mzz}$ resulting from the torsion moment M_{zz}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots a contour of the resultant shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6)

stress_post.plot_stress_mzz_zxy()
```

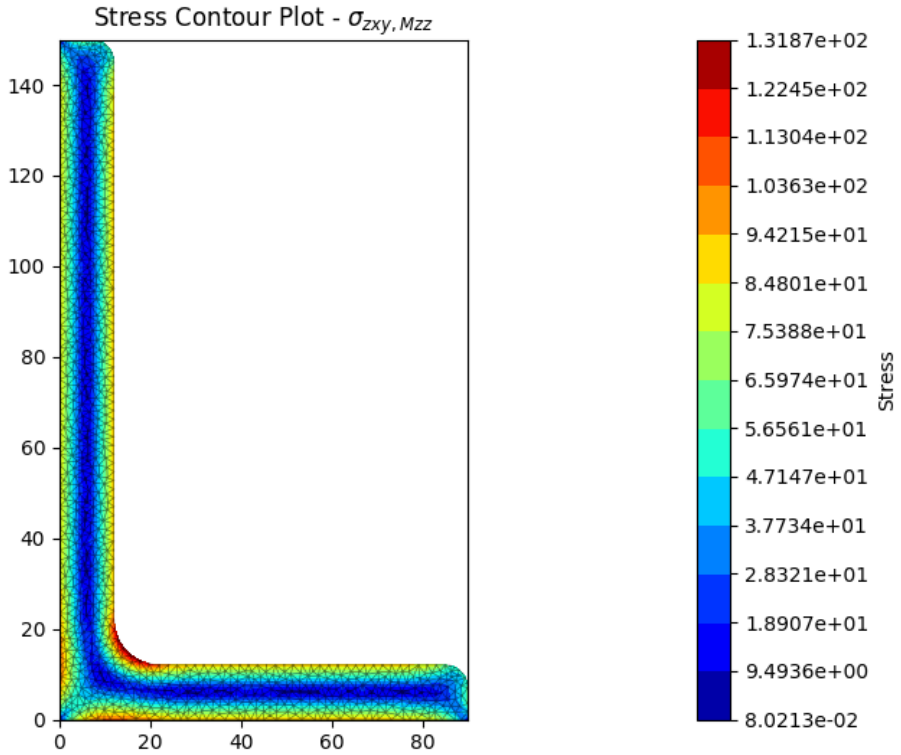


Fig. 11: Contour plot of the shear stress.

StressPost.**plot_vector_mzz_zxy**(*pause=True*)

Produces a vector plot of the resultant shear stress $\sigma_{zxy,Mzz}$ resulting from the torsion moment M_{zz}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example generates a vector plot of the shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6)

stress_post.plot_vector_mzz_zxy()
```

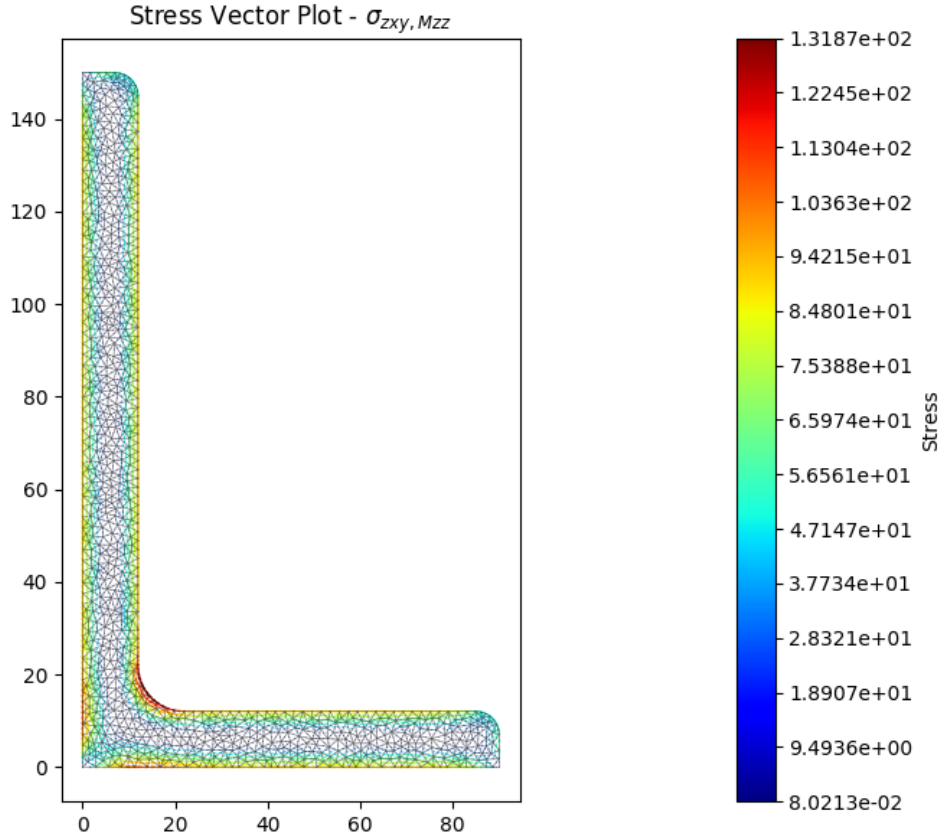


Fig. 12: Vector plot of the shear stress.

Shear Stress ($\sigma_{zx,Vx}$)

StressPost.**plot_stress_vx_zx**(*pause=True*)

Produces a contour plot of the x-component of the shear stress $\sigma_{zx,Vx}$ resulting from the shear force V_x.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the x-component of the shear stress within a 150x90x12 UA section resulting from a shear force in the x-direction of 15 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3)

stress_post.plot_stress_vx_zx()
```

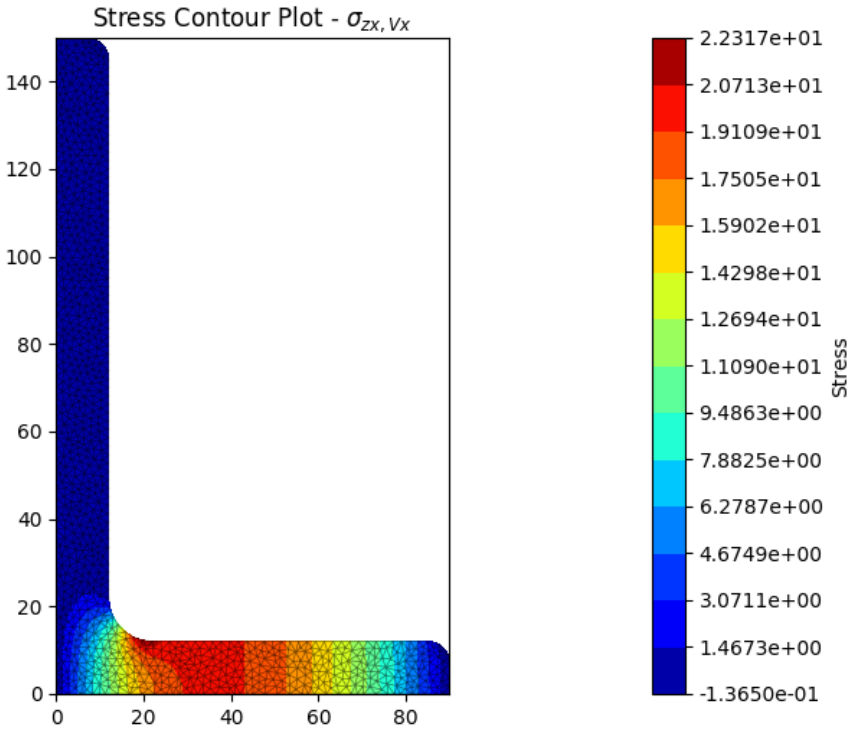


Fig. 13: Contour plot of the shear stress.

Shear Stress ($\sigma_{zy,Vx}$)

StressPost.**plot_stress_vx_zy**(*pause=True*)

Produces a contour plot of the *y*-component of the shear stress $\sigma_{zy,Vx}$ resulting from the shear force V_x.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the y-component of the shear stress within a 150x90x12 UA section resulting from a shear force in the x-direction of 15 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3)

stress_post.plot_stress_vx_zy()
```

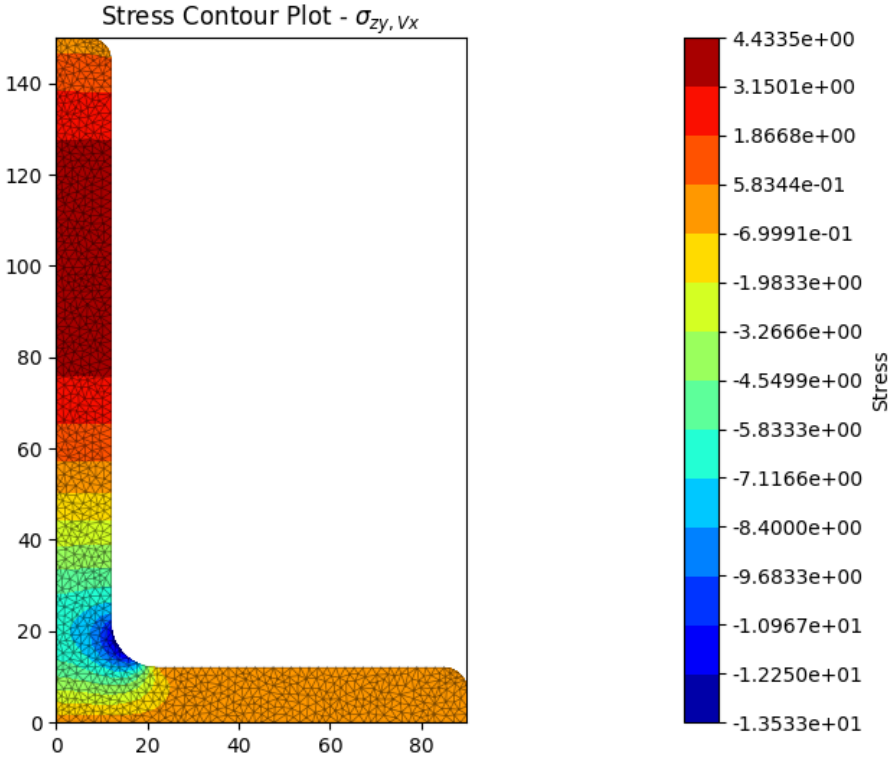


Fig. 14: Contour plot of the shear stress.

Shear Stress ($\sigma_{zxy,Vx}$)

StressPost.**plot_stress_vx_zxy**(*pause=True*)

Produces a contour plot of the resultant shear stress $\sigma_{zxy,Vx}$ resulting from the shear force V_x.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots a contour of the resultant shear stress within a 150x90x12 UA section resulting from a shear force in the x-direction of 15 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3)

stress_post.plot_stress_vx_zxy()
```

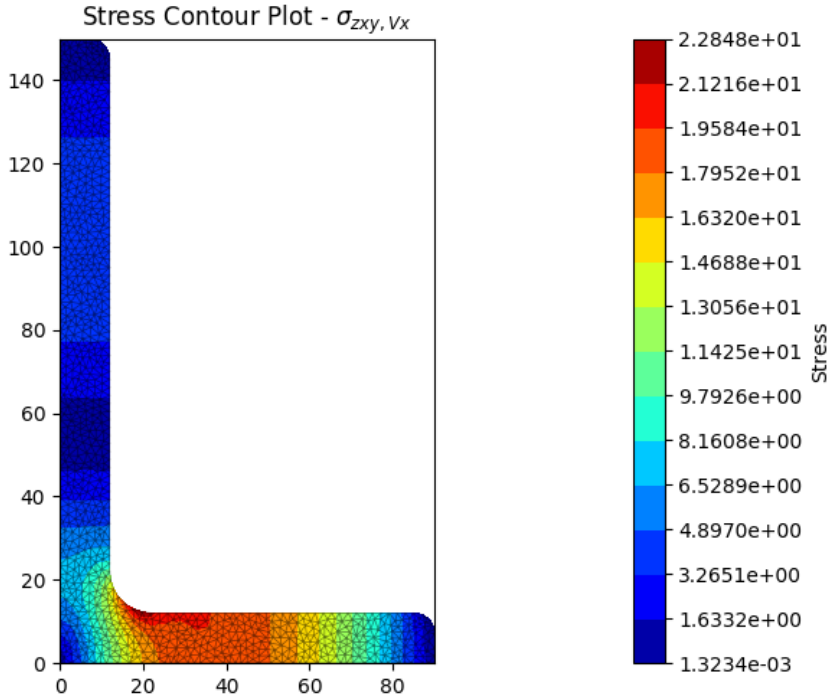


Fig. 15: Contour plot of the shear stress.

StressPost.**plot_vector_vx_zxy**(*pause=True*)

Produces a vector plot of the resultant shear stress $\sigma_{zxy,Vx}$ resulting from the shear force V_x.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example generates a vector plot of the shear stress within a 150x90x12 UA section resulting from a shear force in the x-direction of 15 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3)

stress_post.plot_vector_vx_zxy()
```

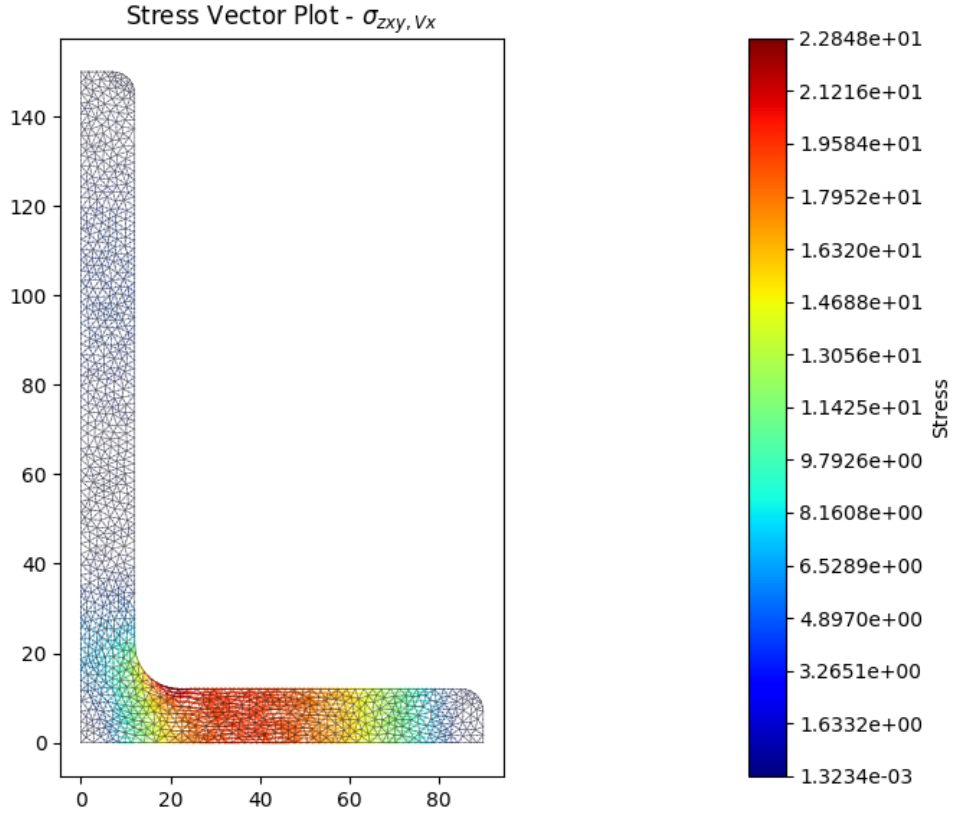


Fig. 16: Vector plot of the shear stress.

Shear Stress ($\sigma_{zx,Vy}$)

StressPost.**plot_stress_vy_zx**(*pause=True*)

Produces a contour plot of the x-component of the shear stress $\sigma_{zx,Vy}$ resulting from the shear force V_y.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the x-component of the shear stress within a 150x90x12 UA section resulting from a shear force in the y-direction of 30 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vy=30e3)

stress_post.plot_stress_vy_zx()
```

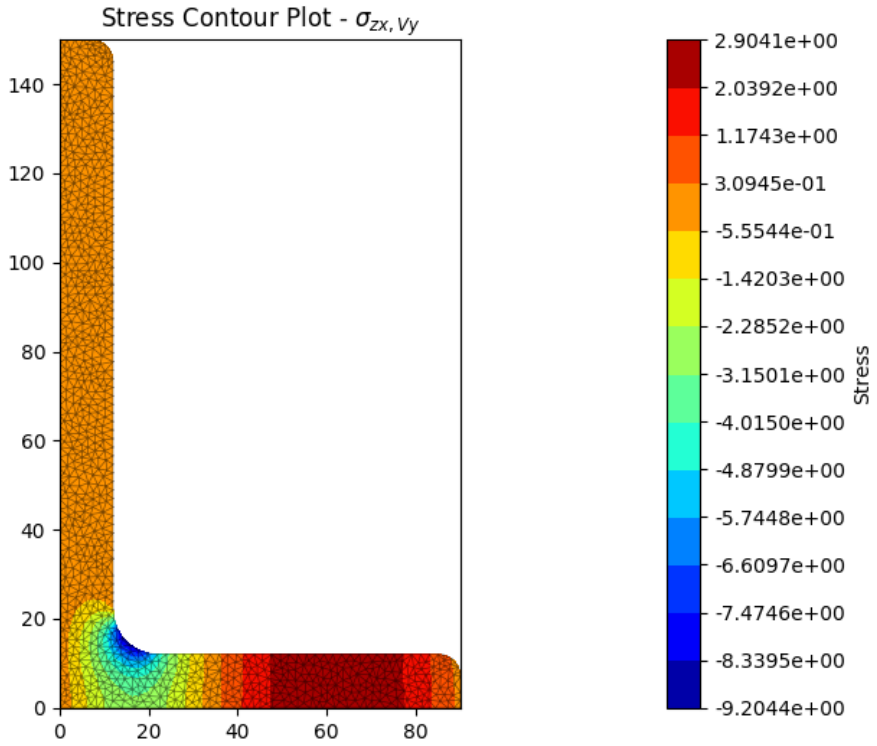


Fig. 17: Contour plot of the shear stress.

Shear Stress ($\sigma_{zy,Vy}$)

StressPost.**plot_stress_vy_zy**(*pause=True*)

Produces a contour plot of the y-component of the shear stress $\sigma_{zy,Vy}$ resulting from the shear force V_y.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the y-component of the shear stress within a 150x90x12 UA section resulting from a shear force in the y-direction of 30 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vy=30e3)

stress_post.plot_stress_vy_zy()
```

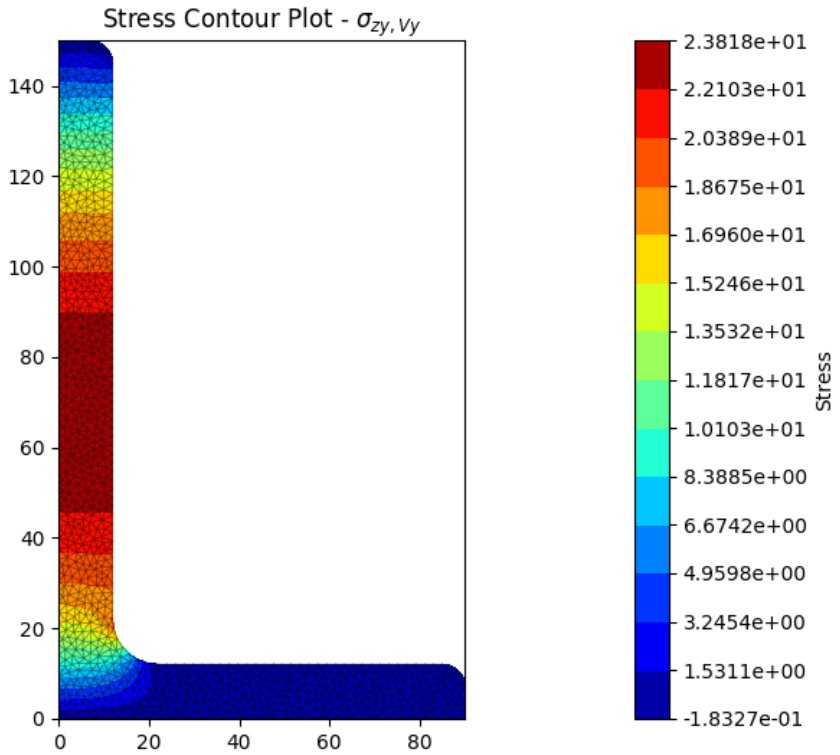


Fig. 18: Contour plot of the shear stress.

Shear Stress ($\sigma_{zxy,Vy}$)

StressPost.**plot_stress_vy_zxy**(*pause=True*)

Produces a contour plot of the resultant shear stress $\sigma_{zxy,Vy}$ resulting from the shear force V_y.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots a contour of the resultant shear stress within a 150x90x12 UA section resulting from a shear force in the y-direction of 30 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vy=30e3)

stress_post.plot_stress_vy_zxy()
```

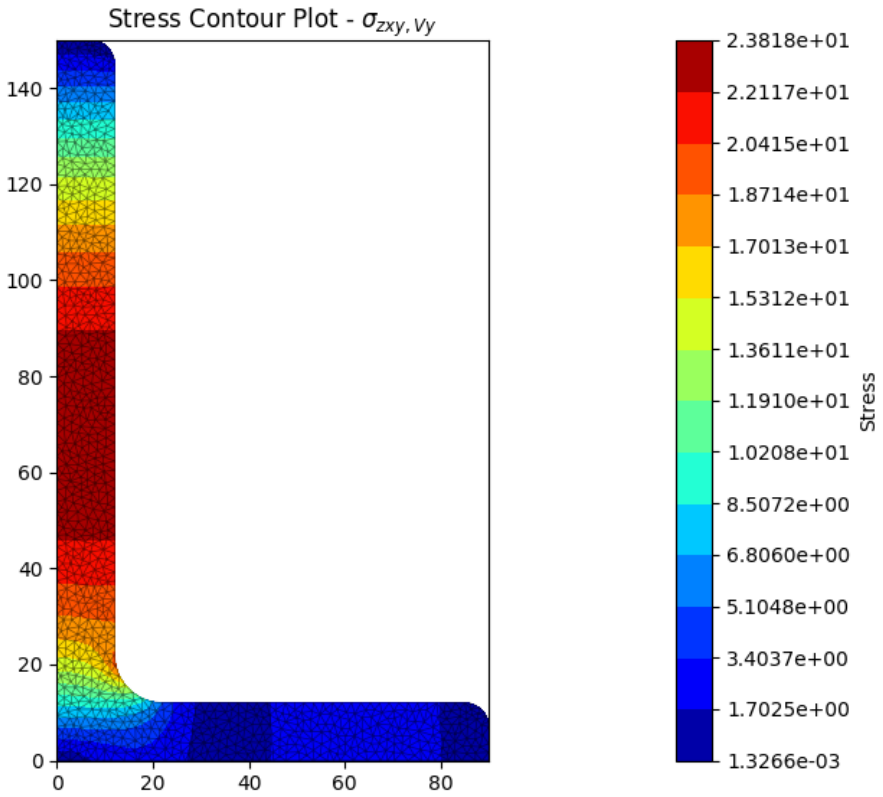


Fig. 19: Contour plot of the shear stress.

StressPost.**plot_vector_vy_zxy**(*pause=True*)

Produces a vector plot of the resultant shear stress $\sigma_{zxy,Vy}$ resulting from the shear force V_y.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example generates a vector plot of the shear stress within a 150x90x12 UA section resulting from a shear force in the y-direction of 30 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vy=30e3)

stress_post.plot_vector_vy_zxy()
```

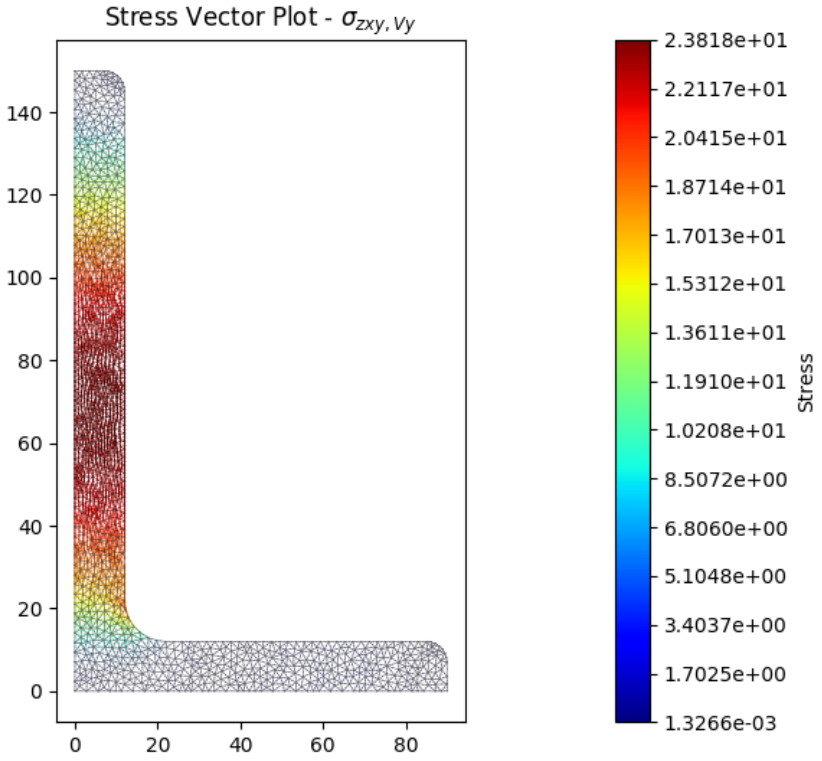


Fig. 20: Vector plot of the shear stress.

Shear Stress ($\sigma_{zx,\Sigma V}$)

`StressPost.`**`plot_stress_v_zx`**(*pause=True*)

Produces a contour plot of the x-component of the shear stress $\sigma_{zx,\Sigma V}$ resulting from the sum of the applied shear forces $V_x + V_y$.

Parameters **`pause`** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (`matplotlib.figure.Figure`, `matplotlib.axes`)

The following example plots the x-component of the shear stress within a 150x90x12 UA section resulting from a shear force of 15 kN in the x-direction and 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3, Vy=30e3)

stress_post.plot_stress_v_zx()
```

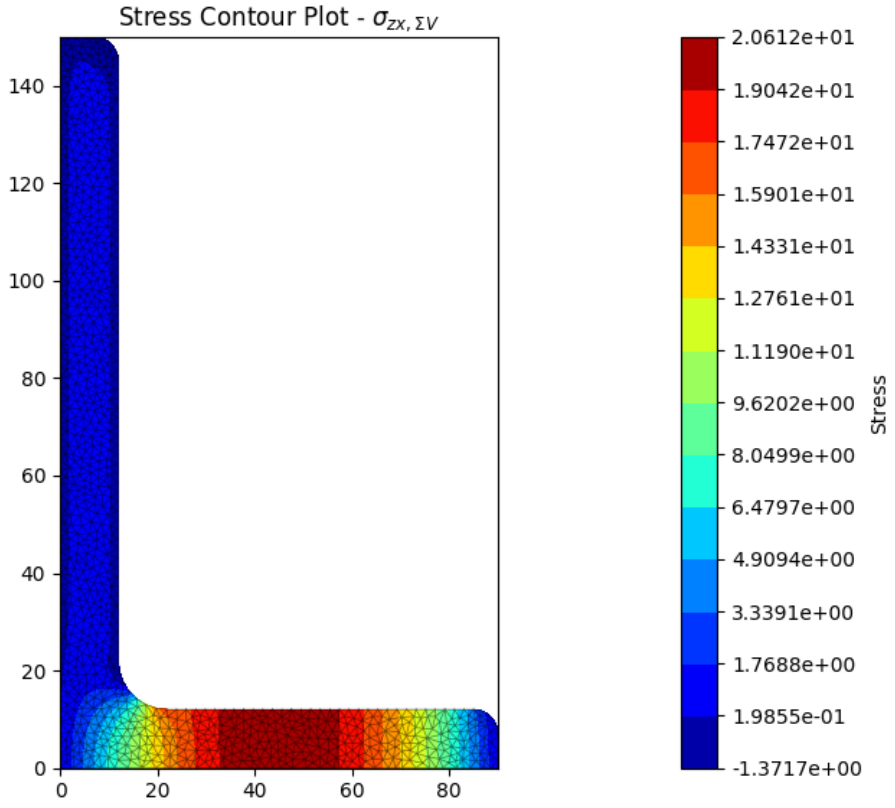


Fig. 21: Contour plot of the shear stress.

Shear Stress ($\sigma_{zy,\Sigma V}$)

`StressPost.`**`plot_stress_v_zy`**(*pause=True*)

Produces a contour plot of the y-component of the shear stress $\sigma_{zy,\Sigma V}$ resulting from the sum of the applied shear forces $V_x + V_y$.

Parameters **`pause`** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (`matplotlib.figure.Figure`, `matplotlib.axes`)

The following example plots the y-component of the shear stress within a 150x90x12 UA section resulting from a shear force of 15 kN in the x-direction and 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3, Vy=30e3)

stress_post.plot_stress_v_zy()
```

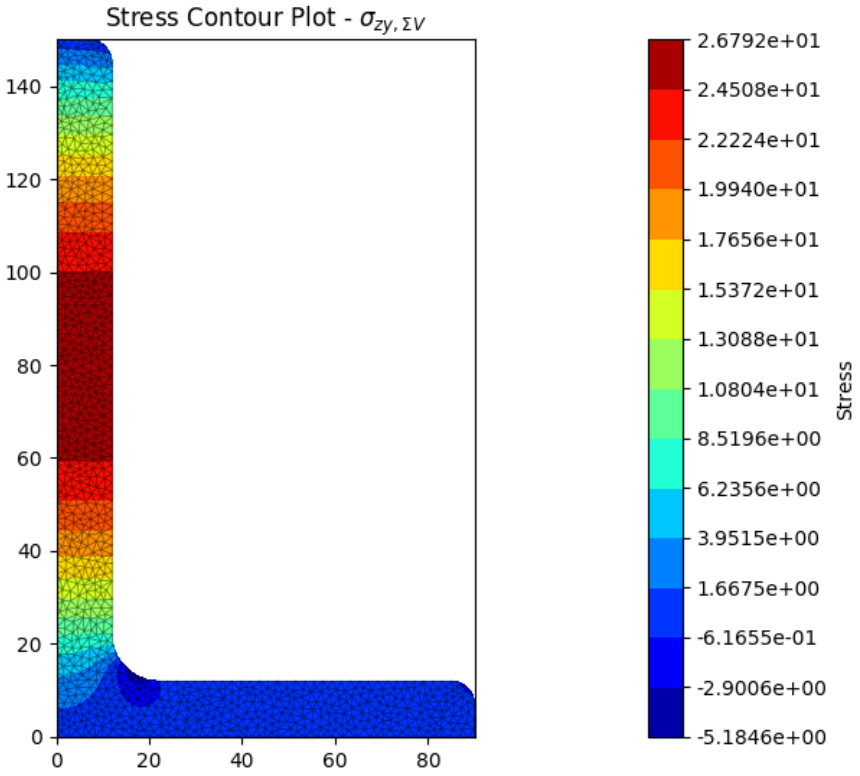


Fig. 22: Contour plot of the shear stress.

Shear Stress ($\sigma_{zxy,\Sigma V}$)

`StressPost.`**`plot_stress_v_zxy`**(*pause=True*)

Produces a contour plot of the resultant shear stress $\sigma_{zxy,\Sigma V}$ resulting from the sum of the applied shear forces $V_x + V_y$.

Parameters **`pause`** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (`matplotlib.figure.Figure`, `matplotlib.axes`)

The following example plots a contour of the resultant shear stress within a 150x90x12 UA section resulting from a shear force of 15 kN in the x-direction and 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3, Vy=30e3)

stress_post.plot_stress_v_zxy()
```

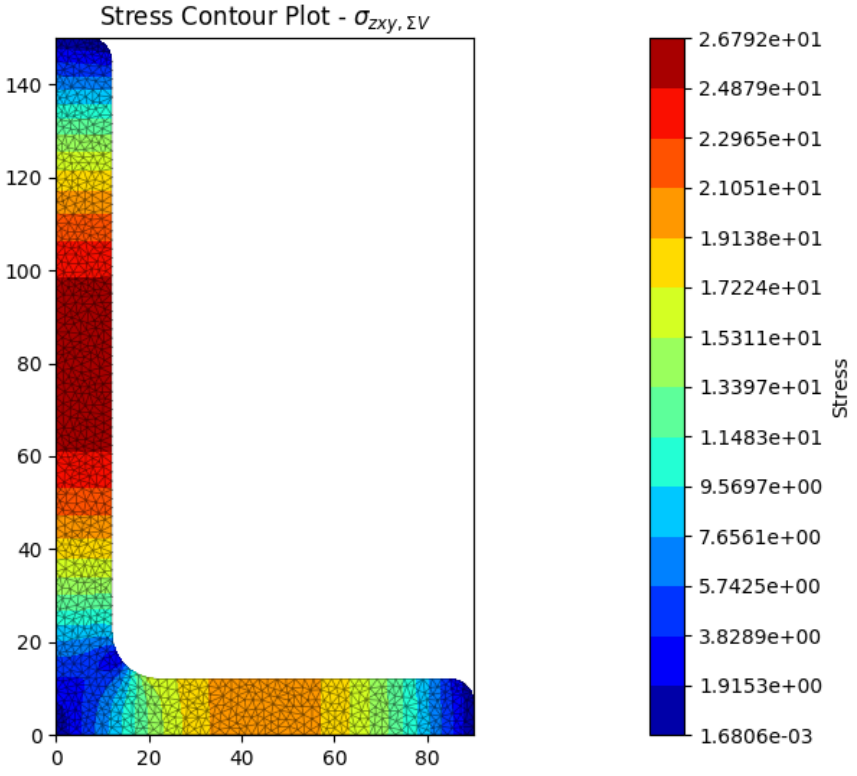


Fig. 23: Contour plot of the shear stress.

StressPost.**plot_vector_v_zxy**(*pause=True*)

Produces a vector plot of the resultant shear stress $\sigma_{zxy,\Sigma V}$ resulting from the sum of the applied shear forces $V_x + V_y$.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example generates a vector plot of the shear stress within a 150x90x12 UA section resulting from a shear force of 15 kN in the x-direction and 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3, Vy=30e3)

stress_post.plot_vector_v_zxy()
```

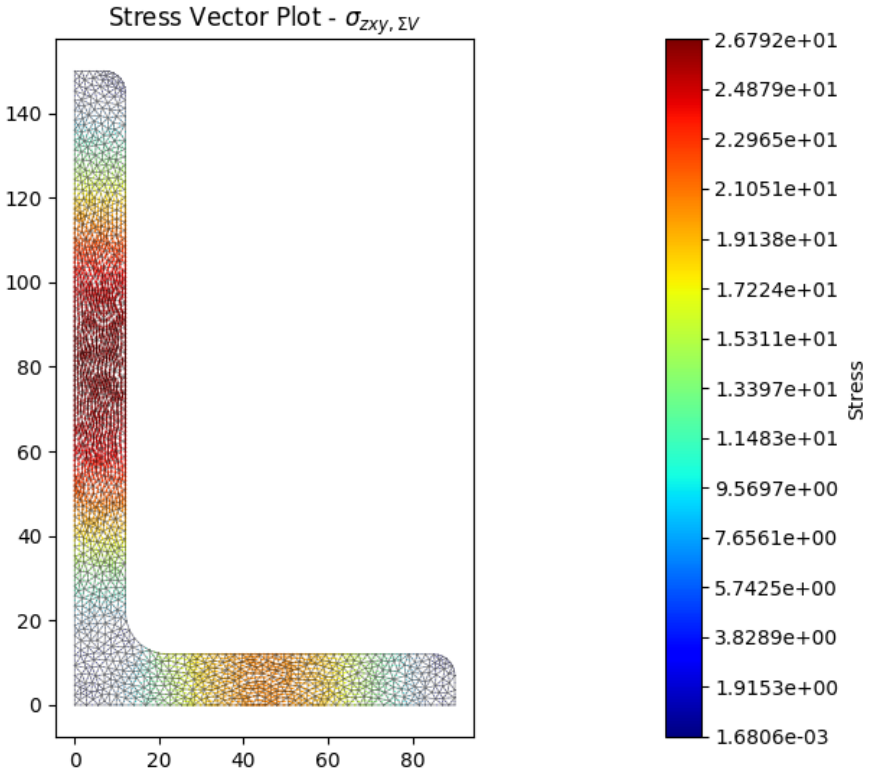


Fig. 24: Vector plot of the shear stress.

5.4.2 Combined Stress Plots

Normal Stress (σ_{zz})

StressPost.**plot_stress_zz**(*pause=True*)

Produces a contour plot of the combined normal stress σ_{zz} resulting from all actions.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the normal stress within a 150x90x12 UA section resulting from an axial force of 100 kN, a bending moment about the x-axis of 5 kN.m and a bending moment about the y-axis of 2 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(N=100e3, Mxx=5e6, Myy=2e6)

stress_post.plot_stress_zz()
```

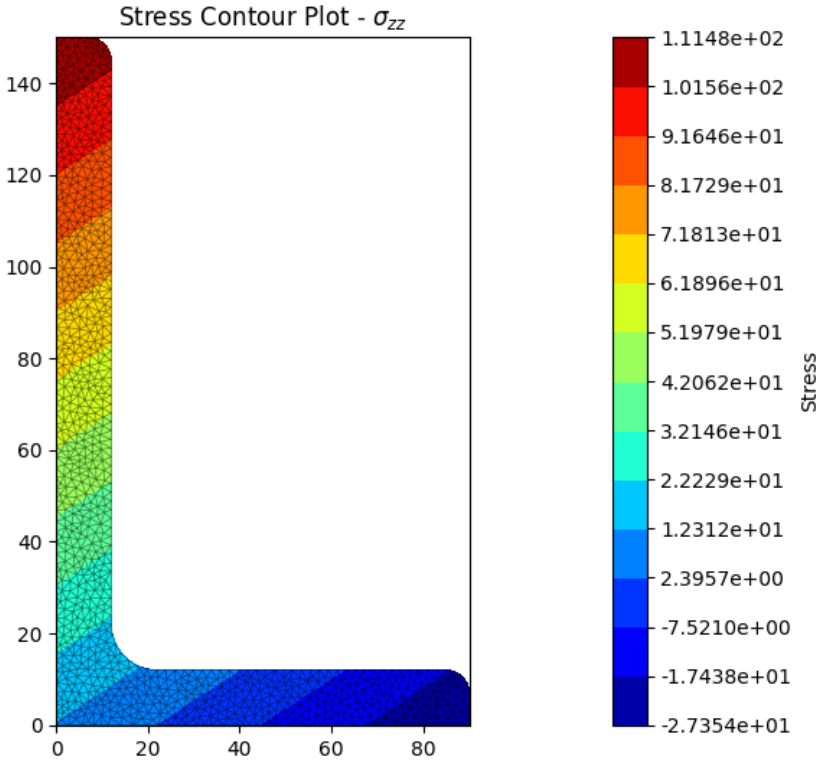


Fig. 25: Contour plot of the normal stress.

Shear Stress (σ_{zx})

StressPost.**plot_stress_zx**(*pause=True*)

Produces a contour plot of the x-component of the shear stress σ_{zx} resulting from all actions.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the x-component of the shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m and a shear force of 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6, Vy=30e3)

stress_post.plot_stress_zx()
```

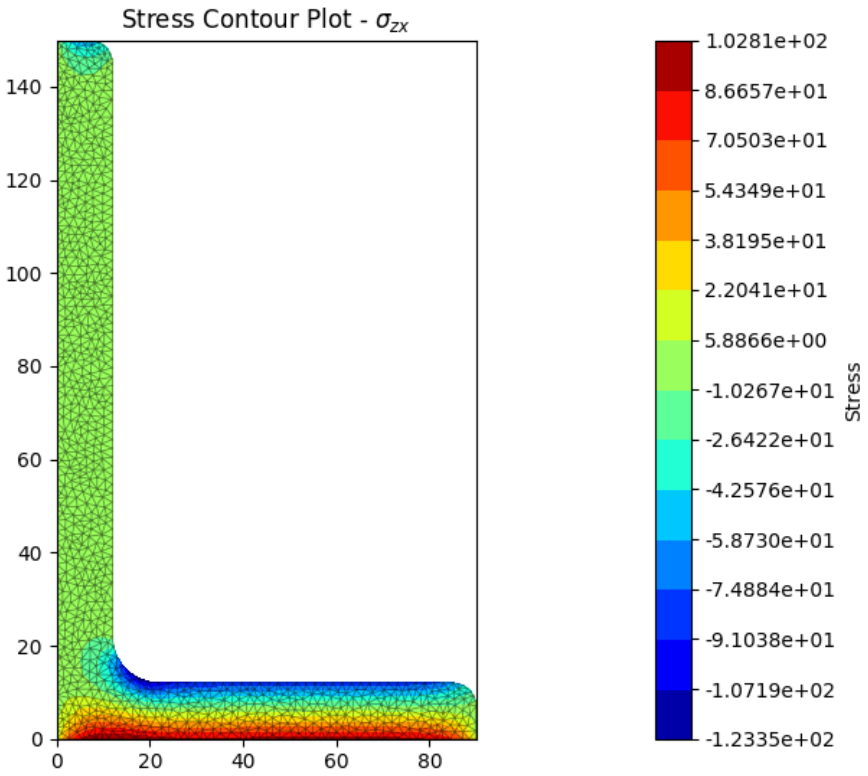


Fig. 26: Contour plot of the shear stress.

Shear Stress (σ_{zy})

StressPost.**plot_stress_zy**(*pause=True*)

Produces a contour plot of the y-component of the shear stress σ_{zy} resulting from all actions.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the y-component of the shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m and a shear force of 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6, Vy=30e3)

stress_post.plot_stress_zy()
```

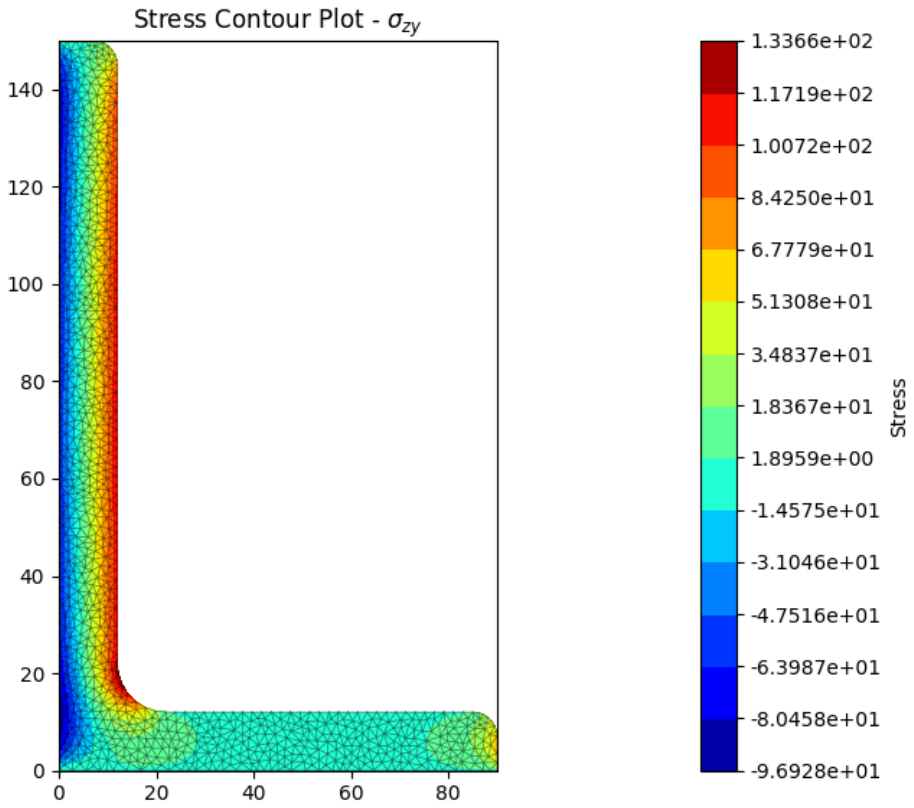


Fig. 27: Contour plot of the shear stress.

Shear Stress (σ_{zxy})

StressPost.**plot_stress_zxy**(*pause=True*)

Produces a contour plot of the resultant shear stress σ_{zxy} resulting from all actions.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots a contour of the resultant shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m and a shear force of 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6, Vy=30e3)

stress_post.plot_stress_zxy()
```

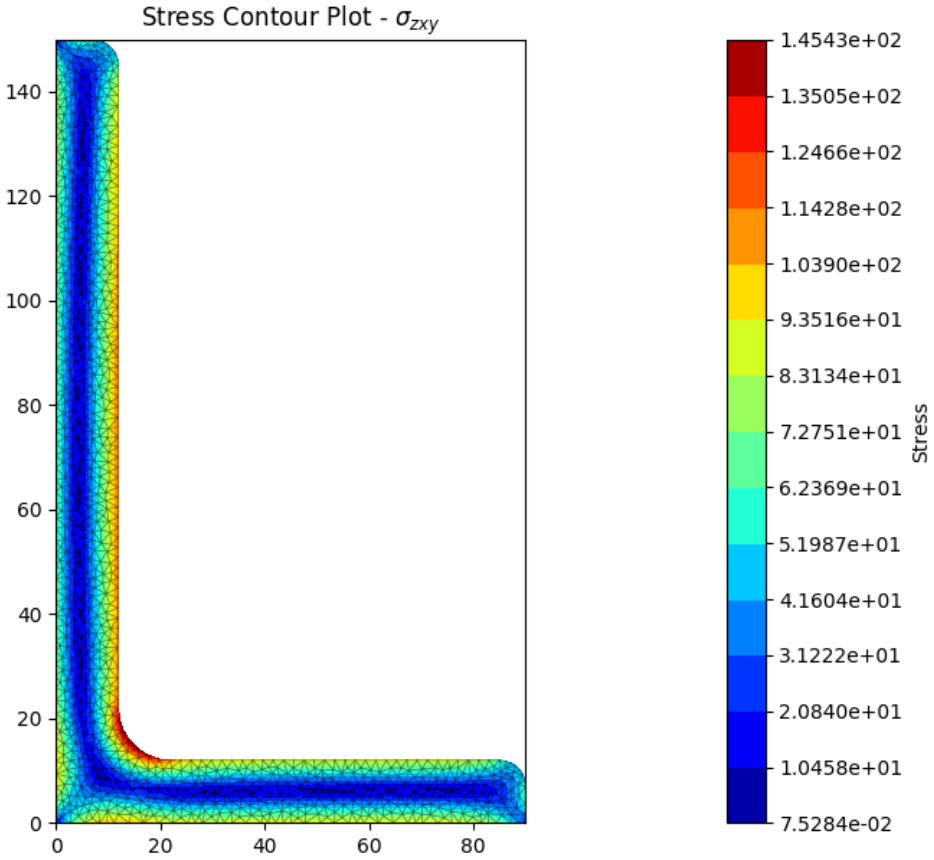


Fig. 28: Contour plot of the shear stress.

StressPost.**plot_vector_zxy**(*pause=True*)

Produces a vector plot of the resultant shear stress σ_{zxy} resulting from all actions.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example generates a vector plot of the shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m and a shear force of 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6, Vy=30e3)

stress_post.plot_vector_zxy()
```

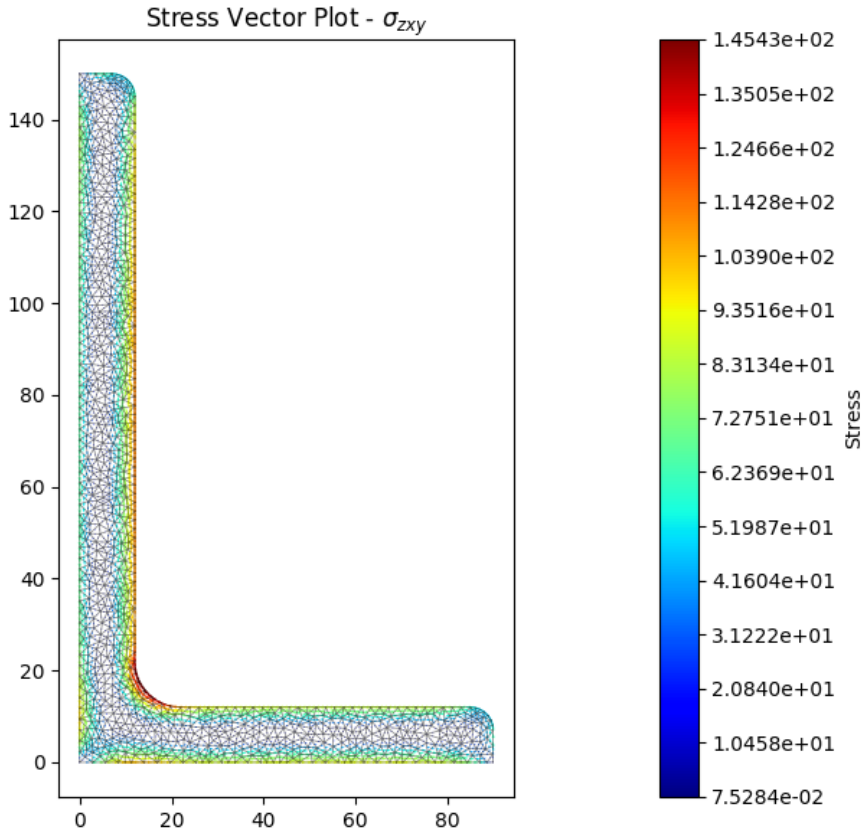


Fig. 29: Vector plot of the shear stress.

von Mises Stress (σ_{vM})

StressPost.**plot_stress_vm**(*pause=True*)

Produces a contour plot of the von Mises stress σ_{vM} resulting from all actions.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots a contour of the von Mises stress within a 150x90x12 UA section resulting from the following actions:

- $N = 50$ kN
- $M_{xx} = -5$ kN.m
- $M_{22} = 2.5$ kN.m
- $M_{zz} = 1.5$ kN.m
- $V_x = 10$ kN
- $V_y = 5$ kN

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(
    N=50e3, Mxx=-5e6, M22=2.5e6, Mzz=0.5e6, Vx=10e3, Vy=5e3
)

stress_post.plot_stress_vm()
```

5.5 Retrieving Cross-Section Stress

All cross-section stresses can be recovered using the *get_stress()* method that belongs to every *StressPost* object:

StressPost.**get_stress**()

Returns the stresses within each material belonging to the current *StressPost* object.

Returns A list of dictionaries containing the cross-section stresses for each material.

Return type list[dict]

A dictionary is returned for each material in the cross-section, containing the following keys and values:

- *'Material'*: Material name
- *'sig_zz_n'*: Normal stress $\sigma_{zz,N}$ resulting from the axial load N
- *'sig_zz_mxx'*: Normal stress $\sigma_{zz,Mxx}$ resulting from the bending moment M_{xx}

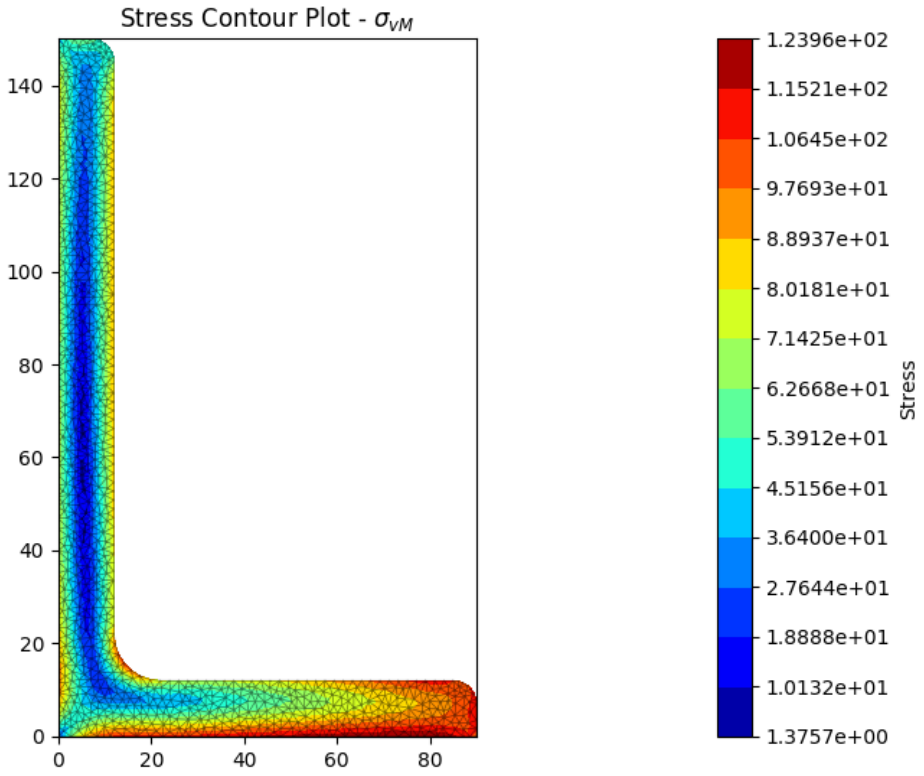


Fig. 30: Contour plot of the von Mises stress.

- '*sig_zz_myy*': Normal stress $\sigma_{zz,Myy}$ resulting from the bending moment M_{yy}
- '*sig_zz_m11*': Normal stress $\sigma_{zz,M11}$ resulting from the bending moment M_{11}
- '*sig_zz_m22*': Normal stress $\sigma_{zz,M22}$ resulting from the bending moment M_{22}
- '*sig_zz_m*': Normal stress $\sigma_{zz,\Sigma M}$ resulting from all bending moments
- '*sig_zx_mzz*': *x*-component of the shear stress $\sigma_{zx,Mzz}$ resulting from the torsion moment
- '*sig_zy_mzz*': *y*-component of the shear stress $\sigma_{zy,Mzz}$ resulting from the torsion moment
- '*sig_zxy_mzz*': Resultant shear stress $\sigma_{zxy,Mzz}$ resulting from the torsion moment
- '*sig_zx_vx*': *x*-component of the shear stress $\sigma_{zx,Vx}$ resulting from the shear force V_x
- '*sig_zy_vx*': *y*-component of the shear stress $\sigma_{zy,Vx}$ resulting from the shear force V_x
- '*sig_zxy_vx*': Resultant shear stress $\sigma_{zxy,Vx}$ resulting from the shear force V_x
- '*sig_zx_vy*': *x*-component of the shear stress $\sigma_{zx,Vy}$ resulting from the shear force V_y
- '*sig_zy_vy*': *y*-component of the shear stress $\sigma_{zy,Vy}$ resulting from the shear force V_y
- '*sig_zxy_vy*': Resultant shear stress $\sigma_{zxy,Vy}$ resulting from the shear force V_y
- '*sig_zx_v*': *x*-component of the shear stress $\sigma_{zx,\Sigma V}$ resulting from all shear forces
- '*sig_zy_v*': *y*-component of the shear stress $\sigma_{zy,\Sigma V}$ resulting from all shear forces
- '*sig_zxy_v*': Resultant shear stress $\sigma_{zxy,\Sigma V}$ resulting from all shear forces
- '*sig_zz*': Combined normal stress σ_{zz} resulting from all actions

- *'sig_zx'*: *x*-component of the shear stress σ_{zx} resulting from all actions
- *'sig_zy'*: *y*-component of the shear stress σ_{zy} resulting from all actions
- *'sig_zxy'*: Resultant shear stress σ_{zxy} resulting from all actions
- *'sig_vm'*: von Mises stress σ_{vM} resulting from all actions

The following example returns the normal stress within a 150x90x12 UA section resulting from an axial force of 10 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(N=10e3)

stresses = stress_post.get_stress()
print('Material: {0}'.format(stresses[0]['Material']))
print('Axial Stresses: {0}'.format(stresses[0]['sig_zz_n']))

$ Material: default
$ Axial Stresses: [3.6402569 3.6402569 3.6402569 ... 3.6402569 3.6402569 3.
↪6402569]
```

CHAPTER 6

Examples

The following examples are located in the `sectionproperties.examples` package.

6.1 Simple Example

The following example calculates the geometric, warping and plastic properties of a 50 mm diameter circle. The circle is discretised with 64 points and a mesh size of 2.5 mm^2.

The geometry and mesh are plotted, and the mesh information printed to the terminal before the analysis is carried out. Detailed time information is printed to the terminal during the cross-section analysis stage. Once the analysis is complete, the cross-section properties are printed to the terminal. The centroidal axis second moments of area and torsion constant are saved to variables and it is shown that, for a circle, the torsion constant is equal to the sum of the second moments of area:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

# create a 50 diameter circle discretised by 64 points
geometry = sections.CircularSection(d=50, n=64)
geometry.plot_geometry()  # plot the geometry

# create a mesh with a mesh size of 2.5
mesh = geometry.create_mesh(mesh_sizes=[2.5])

section = CrossSection(geometry, mesh)  # create a CrossSection object
section.display_mesh_info()  # display the mesh information
section.plot_mesh()  # plot the generated mesh

# perform a geometric, warping and plastic analysis, displaying the time info
section.calculate_geometric_properties(time_info=True)
section.calculate_warping_properties(time_info=True)
section.calculate_plastic_properties(time_info=True)
```

(continues on next page)

(continued from previous page)

```
# print the results to the terminal
section.display_results()

# get the second moments of area and the torsion constant
(ixx_c, iyy_c, ixy_c) = section.get_ic()
j = section.get_j()

# print the sum of the second moments of area and the torsion constant
print("Ixx + Iyy = {0:.3f}".format(ixx_c + iyy_c))
print("J = {0:.3f}".format(j))
```

The following plots are generated by the above example:

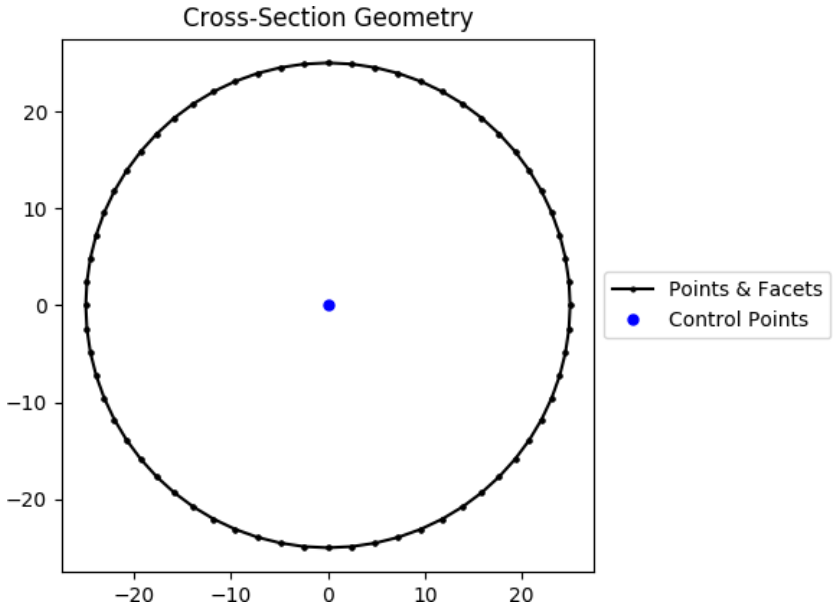


Fig. 1: Circular section geometry.

The following is printed to the terminal:

```
Mesh Statistics:
--2562 nodes
--1247 elements
--1 region

--Calculating geometric section properties...
----completed in 0.765906 seconds---

--Assembing 2562x2562 stiffness matrix and load vector...
----completed in 1.107300 seconds---
--Solving for the warping function using the direct solver...
----completed in 0.023138 seconds---
--Computing the torsion constant...
----completed in 0.000254 seconds---
--Assembling shear function load vectors...
----completed in 1.150968 seconds---
--Solving for the shear functions using the direct solver...
----completed in 0.136840 seconds---
```

(continues on next page)

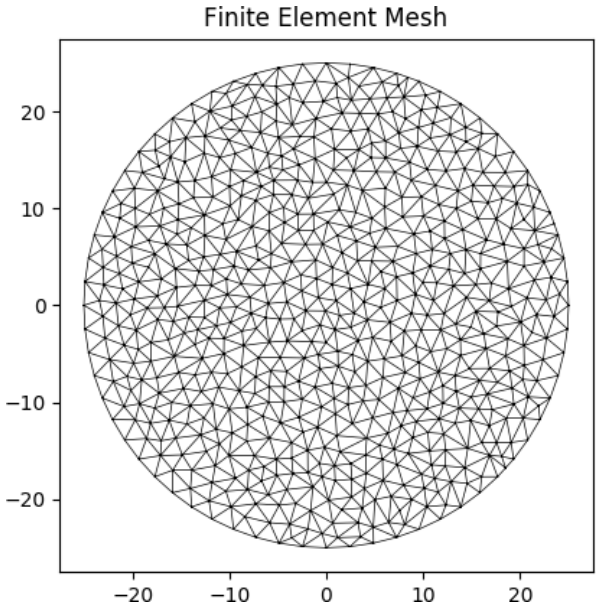


Fig. 2: Mesh generated from the above geometry.

(continued from previous page)

```
--Assembling shear centre and warping moment integrals...
----completed in 0.688029 seconds---
--Calculating shear centres...
----completed in 0.000054 seconds---
--Assembling shear deformation coefficients...
----completed in 1.184013 seconds---
--Assembling monosymmetry integrals...
----completed in 0.784923 seconds---

--Calculating plastic properties...
----completed in 0.669841 seconds---

Section Properties:
A      = 1.960343e+03
Qx     = 1.900702e-13
Qy     = 3.451461e-12
cx     = 1.760642e-15
cy     = 9.695762e-17
Ixx_g  = 3.058119e+05
Iyy_g  = 3.058119e+05
Ixy_g  = -2.785328e-12
Ixx_c  = 3.058119e+05
Iyy_c  = 3.058119e+05
Ixy_c  = -2.785328e-12
Zxx+   = 1.223248e+04
Zxx-   = 1.223248e+04
Zyy+   = 1.223248e+04
Zyy-   = 1.223248e+04
rx     = 1.248996e+01
ry     = 1.248996e+01
phi    = 0.000000e+00
I11_c  = 3.058119e+05
```

(continues on next page)

(continued from previous page)

```
I22_c  = 3.058119e+05
Z11+   = 1.223248e+04
Z11-   = 1.223248e+04
Z22+   = 1.223248e+04
Z22-   = 1.223248e+04
r11    = 1.248996e+01
r22    = 1.248996e+01
J      = 6.116232e+05
Iw     = 4.700106e-02
x_se   = -8.788834e-06
y_se   = -2.644033e-06
x_st   = -8.788834e-06
y_st   = -2.644033e-06
x1_se  = -8.788834e-06
y2_se  = -2.644033e-06
A_sx   = 1.680296e+03
A_sy   = 1.680296e+03
A_s11  = 1.680296e+03
A_s22  = 1.680296e+03
betax+ = -5.288066e-06
betax- = 5.288066e-06
betay+ = -1.757767e-05
betay- = 1.757767e-05
beta11+= -5.288066e-06
beta11-= 5.288066e-06
beta22+= -1.757767e-05
beta22-= 1.757767e-05
x_pc   = 5.313355e-15
y_pc   = 3.649671e-15
Sxx    = 2.078317e+04
Syy    = 2.078317e+04
SF_xx+ = 1.699016e+00
SF_xx- = 1.699016e+00
SF_yy+ = 1.699016e+00
SF_yy- = 1.699016e+00
x11_pc = 5.313355e-15
y22_pc = 3.649671e-15
S11    = 2.078317e+04
S22    = 2.078317e+04
SF_11+ = 1.699016e+00
SF_11- = 1.699016e+00
SF_22+ = 1.699016e+00
SF_22- = 1.699016e+00

Ixx + Iyy = 611623.837
J = 611623.214
```

6.2 Creating a Nastran Section

The following example demonstrates how to create a cross-section defined in a Nastran-based finite element analysis program. The following creates a HAT1 cross-section and calculates the geometric, warping and plastic properties. The HAT1 cross-section is meshed with a maximum elemental area of 0.005.

The geometry and mesh are plotted, and the mesh information printed to the terminal before the analysis is carried out. Detailed time information is printed to the terminal during the cross-section analysis stage. Once the analysis

is complete, the cross-section properties are printed to the terminal. The centroidal axis second moments of area and torsion constant are saved to variables and it is shown that, for non-circular sections, the torsion constant is not equal to the sum of the second moments of area:

```
import sectionproperties.pre.nastran_sections as nsections
from sectionproperties.analysis.cross_section import CrossSection

# create a HAT1 section
geometry = nsections.HAT1Section(DIM1=4.0, DIM2=2.0, DIM3=1.5, DIM4=0.1875, DIM5=0.
↪375)
geometry.plot_geometry()  # plot the geometry

# create a mesh with a maximum elemental area of 0.005
mesh = geometry.create_mesh(mesh_sizes=[0.005])

section = CrossSection(geometry, mesh)  # create a CrossSection object
section.display_mesh_info()  # display the mesh information
section.plot_mesh()  # plot the generated mesh`

# perform a geometric, warping and plastic anaylsis, displaying the time info
section.calculate_geometric_properties(time_info=True)
section.calculate_warping_properties(time_info=True)
section.calculate_plastic_properties(time_info=True)

# print the results to the terminal
section.display_results()

# get the second moments of area and the torsion constant
(ixx_c, iyy_c, ixy_c) = section.get_ic()
j = section.get_j()

# print the sum of the second moments of area and the torsion constant
print("Ixx + Iyy = {0:.3f}".format(ixx_c + iyy_c))
print("J = {0:.3f}".format(j))
```

The following plots are generated by the above example:

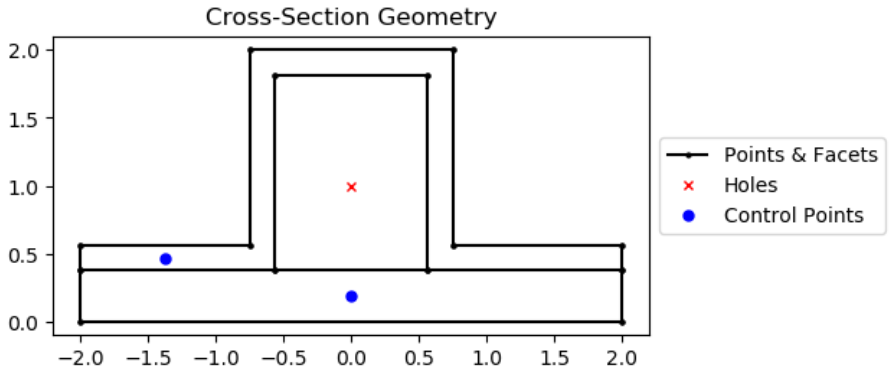


Fig. 3: Circular section geometry.

The following is printed to the terminal:

```
Mesh Statistics:
--2038 nodes
--926 elements
```

(continues on next page)

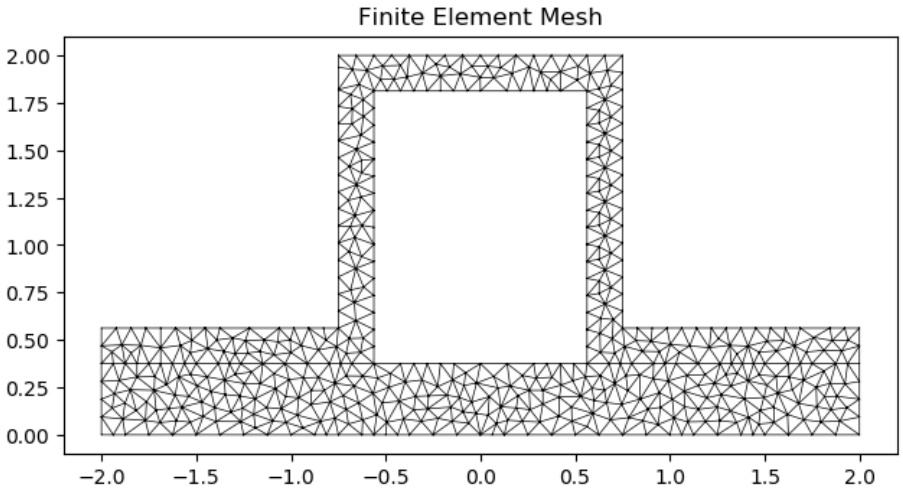


Fig. 4: Mesh generated from the above geometry.

(continued from previous page)

```
--2 regions

--Calculating geometric section properties...
----completed in 0.367074 seconds---

--Assembing 2038x2038 stiffness matrix and load vector...
----completed in 0.515934 seconds---
--Solving for the warping function using the direct solver...
----completed in 0.005604 seconds---
--Computing the torsion constant...
----completed in 0.000104 seconds---
--Assembling shear function load vectors...
----completed in 0.525532 seconds---
--Solving for the shear functions using the direct solver...
----completed in 0.064247 seconds---
--Assembling shear centre and warping moment integrals...
----completed in 0.331969 seconds---
--Calculating shear centres...
----completed in 0.000043 seconds---
--Assembling shear deformation coefficients...
----completed in 0.511631 seconds---
--Assembling monosymmetry integrals...
----completed in 0.389498 seconds---

--Calculating plastic properties...
----completed in 0.131321 seconds---

Section Properties:
A    = 2.789062e+00
Qx   = 1.626709e+00
Qy   = -1.424642e-16
cx   = -5.107959e-17
cy   = 5.832458e-01
Ixx_g        = 1.935211e+00
Iyy_g        = 3.233734e+00
Ixy_g        = -1.801944e-16
```

(continues on next page)

(continued from previous page)

```
Ixx_c         = 9.864400e-01
Iyy_c         = 3.233734e+00
Ixy_c         = -9.710278e-17
Zxx+          = 6.962676e-01
Zxx-          = 1.691294e+00
Zyy+          = 1.616867e+00
Zyy-          = 1.616867e+00
rx   = 5.947113e-01
ry   = 1.076770e+00
phi  = -9.000000e+01
I11_c         = 3.233734e+00
I22_c         = 9.864400e-01
Z11+          = 1.616867e+00
Z11-          = 1.616867e+00
Z22+          = 1.691294e+00
Z22-          = 6.962676e-01
r11  = 1.076770e+00
r22  = 5.947113e-01
J    = 9.878443e-01
Iw   = 1.160810e-01
x_se          = 4.822719e-05
y_se          = 4.674792e-01
x_st          = 4.822719e-05
y_st          = 4.674792e-01
x1_se         = 1.157666e-01
y2_se         = 4.822719e-05
A_sx          = 1.648312e+00
A_sy          = 6.979733e-01
A_s11         = 6.979733e-01
A_s22         = 1.648312e+00
betax+        = -2.746928e-01
betax-        = 2.746928e-01
betay+        = 9.645438e-05
betay-        = -9.645438e-05
beta11+       = 9.645438e-05
beta11-       = -9.645438e-05
beta22+       = 2.746928e-01
beta22-       = -2.746928e-01
x_pc          = -5.107959e-17
y_pc          = 3.486328e-01
Sxx  = 1.140530e+00
Syy  = 2.603760e+00
SF_xx+        = 1.638062e+00
SF_xx-        = 6.743533e-01
SF_yy+        = 1.610373e+00
SF_yy-        = 1.610373e+00
x11_pc        = -3.671369e-17
y22_pc        = 3.486328e-01
S11  = 2.603760e+00
S22  = 1.140530e+00
SF_11+        = 1.610374e+00
SF_11-        = 1.610374e+00
SF_22+        = 6.743539e-01
SF_22-        = 1.638064e+00

Ixx + Iyy = 4.220
J = 0.988
```

6.3 Creating Custom Geometry

The following example demonstrates how geometry objects can be created from a list of points, facets, holes and control points. An straight angle section with a plate at its base is created from a list of points and facets. The bottom plate is assigned a separate control point meaning two discrete regions are created. Creating separate regions allows the user to control the mesh size in each region and assign material properties to different regions. The geometry is cleaned to remove the overlapping facet at the junction of the angle and the plate. A geometric, warping and plastic analysis is then carried out.

The geometry and mesh are plotted before the analysis is carried out. Once the analysis is complete, a plot of the various calculated centroids is generated:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

# define parameters for the angle section
a = 1
b = 2
t = 0.1

# build the lists of points, facets, holes and control points
points = [[-t/2, -2*a], [t/2, -2*a], [t/2, -t/2], [a, -t/2], [a, t/2],
          [-t/2, t/2], [-b/2, -2*a], [b/2, -2*a], [b/2, -2*a-t],
          [-b/2, -2*a-t]]
facets = [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 0], [6, 7], [7, 8],
          [8, 9], [9, 6]]
holes = []
control_points = [[0, 0], [0, -2*a-t/2]]

# create the custom geometry object
geometry = sections.CustomSection(points, facets, holes, control_points)
geometry.clean_geometry()  # clean the geometry
geometry.plot_geometry()  # plot the geometry

# create the mesh - use a smaller refinement for the angle region
mesh = geometry.create_mesh(mesh_sizes=[0.0005, 0.001])

# create a CrossSection object
section = CrossSection(geometry, mesh)
section.plot_mesh()  # plot the generated mesh

# perform a geometric, warping and plastic analysis
section.calculate_geometric_properties()
section.calculate_warping_properties()
section.calculate_plastic_properties()

# plot the centroids
section.plot_centroids()
```

The following plots are generated by the above example:

6.4 Creating a Merged Section

The following example demonstrates how to merge multiple geometry objects into a single geometry object. A 150x100x6 RHS is modelled with a solid 50x50 triangular section on its top and a 100x100x6 EA section on its

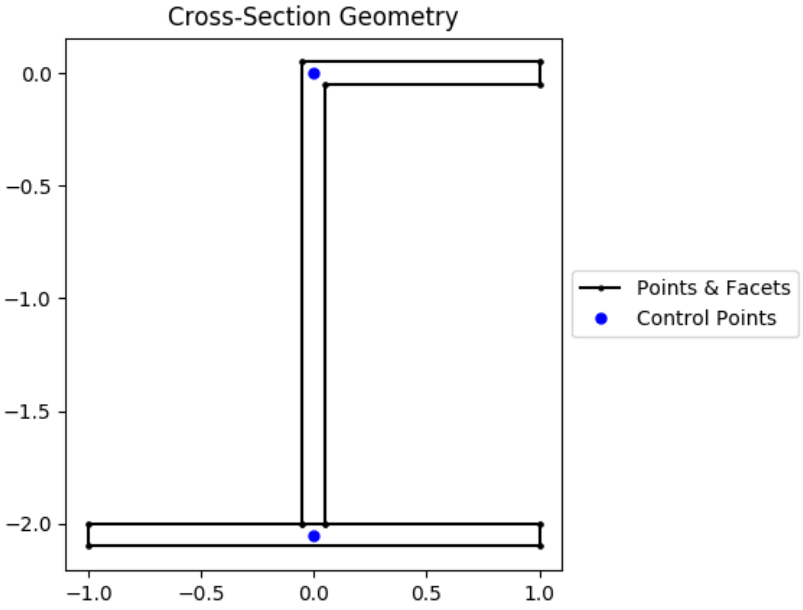


Fig. 5: Plot of the generated geometry object.

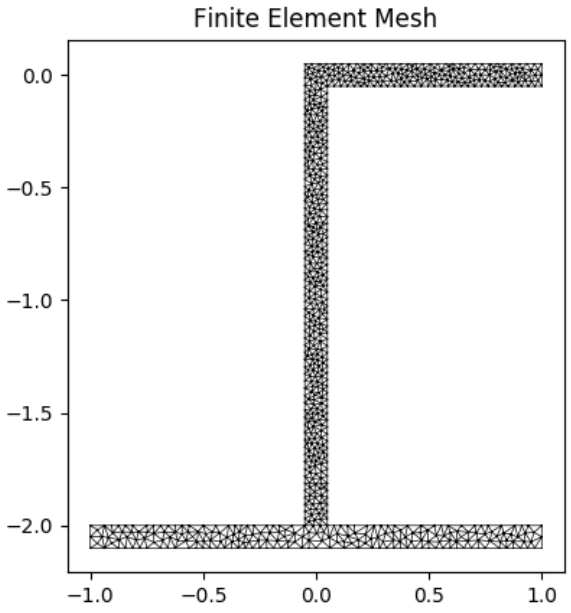


Fig. 6: Mesh generated from the above geometry.

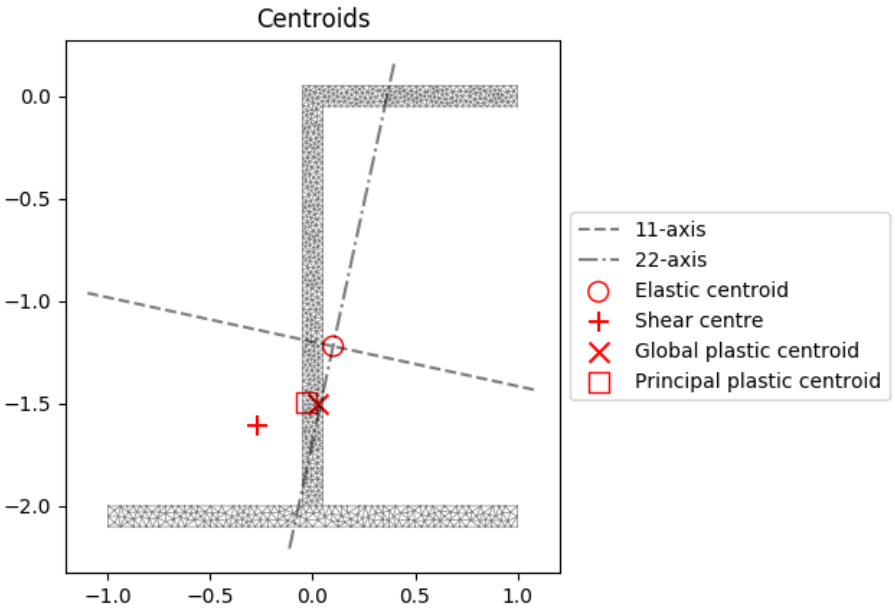


Fig. 7: Plot of the centroids and the principal axis.

right side. The three geometry objects are merged together using the *MergedSection* class. The order of the geometry objects in the list that is passed into the constructor of the *MergedSection* class is important, as this same order relates to specifying mesh sizes and material properties.

Once the geometry has been merged, it is vital to clean the geometry to remove any artefacts that may impede the meshing algorithm. A mesh is created with a mesh size of 2.5 mm^2 for the RHS (first in `section_list`), 5 mm^2 for the triangle (second in `section_list`) and 3 mm^2 for the angle (last in `section_list`).

The geometry and mesh are plotted, and the mesh information printed to the terminal before the analysis is carried out. Detailed time information is printed to the terminal during the cross-section analysis stage. Once the analysis is complete, the centroids are plotted:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

# create a 150x100x6 RHS
rhs = sections.Rhs(d=150, b=100, t=6, r_out=15, n_r=8)

# create a triangular section on top of the RHS
points = [[0, 0], [50, 0], [25, 50]]
facets = [[0, 1], [1, 2], [2, 0]]
holes = []
control_points = [[25, 25]]
triangle = sections.CustomSection(points, facets, holes, control_points,
                                  shift=[25, 150])

# create a 100x100x6 EA on the right of the RHS
angle = sections.AngleSection(d=100, b=100, t=6, r_r=8, r_t=5, n_r=8,
                              shift=[100, 25])

# create a list of the sections to be merged
section_list = [rhs, triangle, angle]
```

(continues on next page)

(continued from previous page)

```
# merge the three sections into one geometry object
geometry = sections.MergedSection(section_list)

# clean the geometry - print cleaning information to the terminal
geometry.clean_geometry(verbose=True)
geometry.plot_geometry()  # plot the geometry

# create a mesh - use a mesh size of 2.5 for the RHS, 5 for the triangle and
# 3 for the angle
mesh = geometry.create_mesh(mesh_sizes=[2.5, 5, 3])

# create a CrossSection object
section = CrossSection(geometry, mesh)
section.display_mesh_info()  # display the mesh information
section.plot_mesh()  # plot the generated mesh

# perform a geometric, warping and plastic analysis, displaying the time info
# and the iteration info for the plastic analysis
section.calculate_geometric_properties(time_info=True)
section.calculate_warping_properties(time_info=True)
section.calculate_plastic_properties(time_info=True, verbose=True)

# plot the centroids
section.plot_centroids()
```

The following plots are generated by the above example:

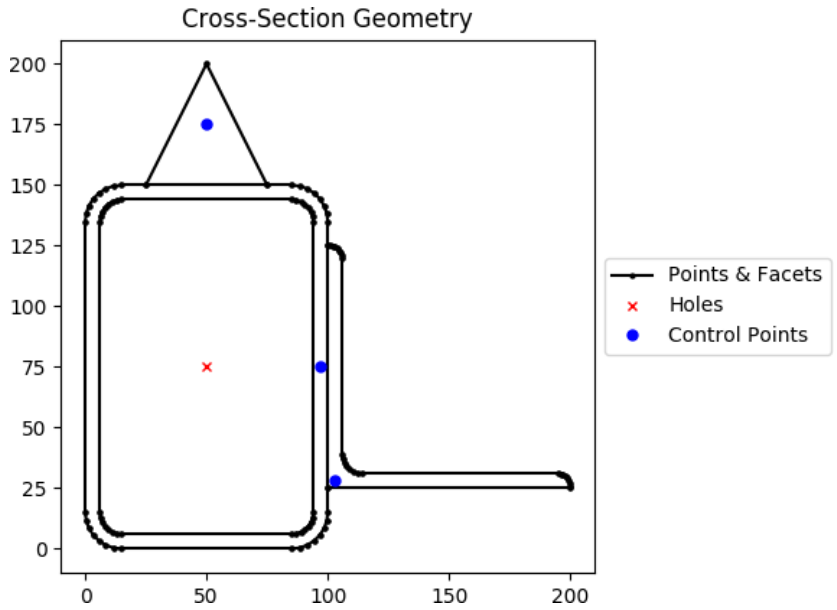


Fig. 8: Plot of the generated geometry object.

The following is printed to the terminal:

```
Removed overlapping facets... Rebuilt with points: [30, 67, 93, 32]
Removed overlapping facets... Rebuilt with points: [46, 65, 64, 48]
Mesh Statistics:
--6053 nodes
```

(continues on next page)

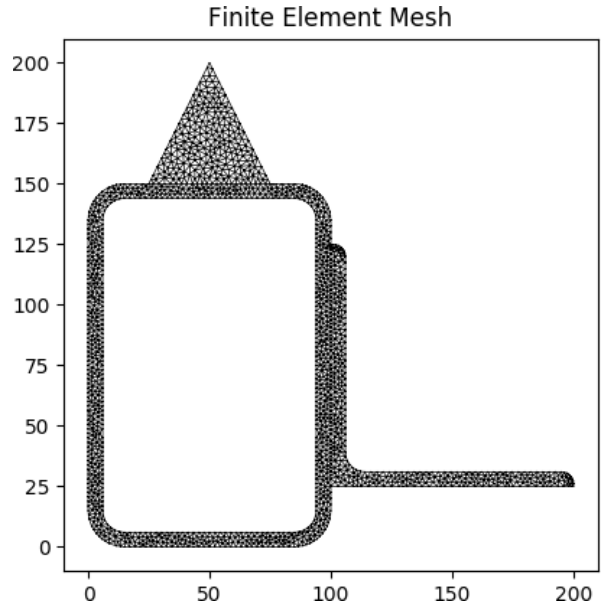


Fig. 9: Mesh generated from the above geometry.

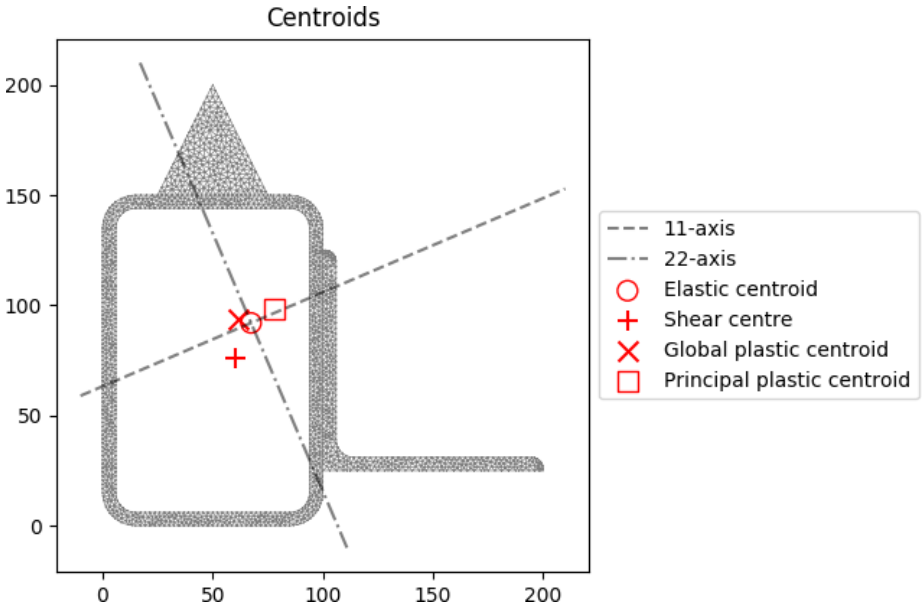


Fig. 10: Plot of the centroids and the principal axis.

(continued from previous page)

```
--2755 elements
--3 regions

--Calculating geometric section properties...
----completed in 1.730845 seconds---

--Assembing 6053x6053 stiffness matrix and load vector...
----completed in 2.793801 seconds---
--Solving for the warping function using the direct solver...
----completed in 0.021323 seconds---
--Computing the torsion constant...
----completed in 0.000316 seconds---
--Assembling shear function load vectors...
----completed in 2.552404 seconds---
--Solving for the shear functions using the direct solver...
----completed in 0.604847 seconds---
--Assembling shear centre and warping moment integrals...
----completed in 1.578075 seconds---
--Calculating shear centres...
----completed in 0.000068 seconds---
--Assembling shear deformation coefficients...
----completed in 2.438405 seconds---

--Calculating plastic properties...
---x-axis plastic centroid calculation converged at 1.66608e+00 in 7 iterations.
---y-axis plastic centroid calculation converged at -5.83761e+00 in 10 iterations.
---11-axis plastic centroid calculation converged at -1.43134e+00 in 7 iterations.
---22-axis plastic centroid calculation converged at -1.21319e+01 in 9 iterations.
----completed in 2.710146 seconds---
```

6.5 Mirroring and Rotating Geometry

The following example demonstrates how geometry objects can be mirrored and rotated. A 200PFC and 150PFC are placed back-to-back by using the *mirror_section()* method and are rotated counter-clockwise by 30 degrees by using the *rotate_section()* method. The geometry is cleaned to ensure there are no overlapping facets along the junction between the two PFCs. A geometric, warping and plastic analysis is then carried out.

The geometry and mesh are plotted, and the mesh information printed to the terminal before the analysis is carried out. Detailed time information is printed to the terminal during the cross-section analysis stage and iteration information printed for the plastic analysis. Once the analysis is complete, a plot of the various calculated centroids is generated:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

# create a 200PFC and a 150PFC
pfc1 = sections.PfcSection(d=203, b=133, t_f=7.8, t_w=5.8, r=8.9, n_r=8)
pfc2 = sections.PfcSection(d=150, b=133, t_f=7.8, t_w=5.8, r=8.9, n_r=8,
                           shift=[0, 26.5])

# mirror the 200 PFC about the y-axis
pfc1.mirror_section(axis='y', mirror_point=[0, 0])

# merge the pfc sections
geometry = sections.MergedSection([pfc1, pfc2])
```

(continues on next page)

(continued from previous page)

```
# rotate the geometry counter-clockwise by 30 degrees
geometry.rotate_section(angle=30)

# clean the geometry - print cleaning information to the terminal
geometry.clean_geometry(verbose=True)
geometry.plot_geometry()  # plot the geometry

# create a mesh - use a mesh size of 5 for the 200PFC and 4 for the 150PFC
mesh = geometry.create_mesh(mesh_sizes=[5, 4])

# create a CrossSection object
section = CrossSection(geometry, mesh)
section.display_mesh_info()  # display the mesh information
section.plot_mesh()  # plot the generated mesh

# perform a geometric, warping and plastic analysis, displaying the time info
# and the iteration info for the plastic analysis
section.calculate_geometric_properties(time_info=True)
section.calculate_warping_properties(time_info=True)
section.calculate_plastic_properties(time_info=True, verbose=True)

# plot the centroids
section.plot_centroids()
```

The following plots are generated by the above example:

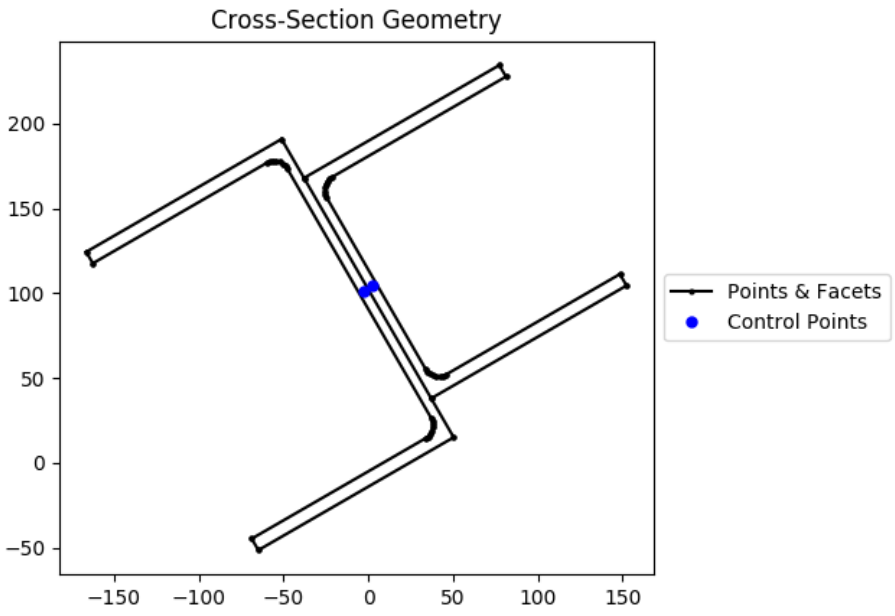


Fig. 11: Plot of the generated geometry object.

The following is printed to the terminal:

```
Removed overlapping facets... Rebuilt with points: [21, 43, 22, 0]
Mesh Statistics:
--4841 nodes
```

(continues on next page)

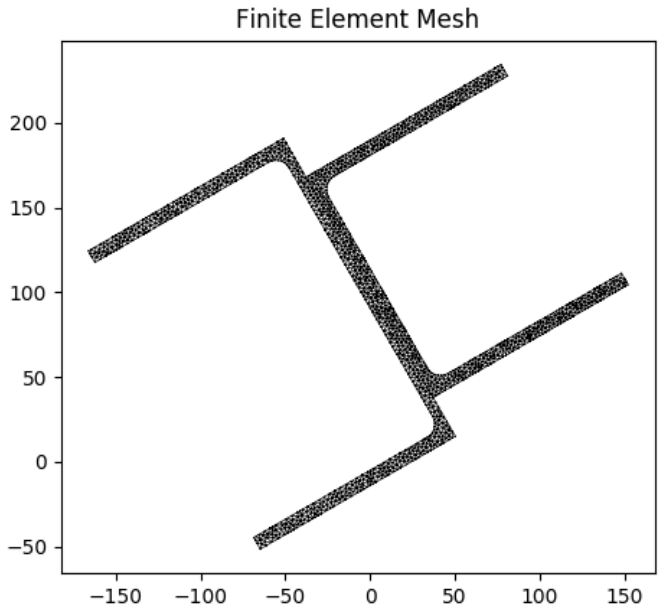


Fig. 12: Mesh generated from the above geometry.

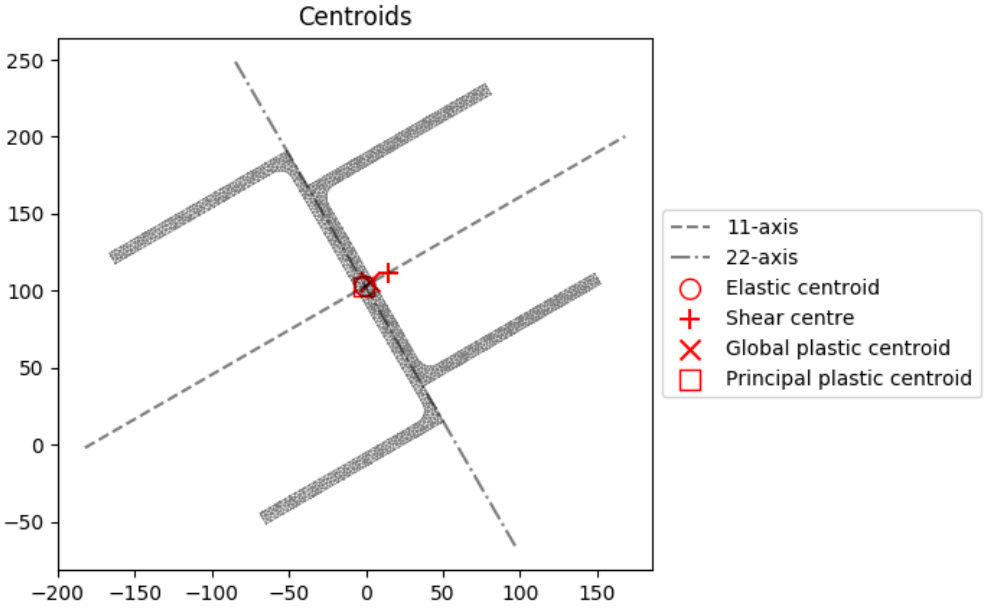


Fig. 13: Plot of the centroids and the principal axis.

(continued from previous page)

```
--2152 elements
--2 regions

--Calculating geometric section properties...
----completed in 1.350236 seconds---

--Assembing 4841x4841 stiffness matrix and load vector...
----completed in 2.002365 seconds---
--Solving for the warping function using the direct solver...
----completed in 0.013307 seconds---
--Computing the torsion constant...
----completed in 0.000222 seconds---
--Assembling shear function load vectors...
----completed in 1.910170 seconds---
--Solving for the shear functions using the direct solver...
----completed in 0.623121 seconds---
--Assembling shear centre and warping moment integrals...
----completed in 1.163591 seconds---
--Calculating shear centres...
----completed in 0.000059 seconds---
--Assembling shear deformation coefficients...
----completed in 1.831169 seconds---

--Calculating plastic properties...
---x-axis plastic centroid calculation converged at 2.77651e+00 in 9 iterations.
---y-axis plastic centroid calculation converged at 3.02247e+00 in 5 iterations.
---11-axis plastic centroid calculation converged at -2.41585e-13 in 3 iterations.
---22-axis plastic centroid calculation converged at 6.10669e-01 in 5 iterations.
----completed in 0.860817 seconds---
```

6.6 Performing a Stress Analysis

The following example demonstrates how a stress analysis can be performed on a cross-section. A 150x100x6 RHS is modelled on its side with a maximum mesh area of 2 mm^2. The pre-requisite geometric and warping analyses are performed before two separate stress analyses are undertaken. The first combines bending and shear about the x-axis with a torsion moment and the second combines bending and shear about the y-axis with a torsion moment.

After the analysis is performed, various plots of the stresses are generated:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

# create a 150x100x6 RHS on its side
geometry = sections.Rhs(d=100, b=150, t=6, r_out=15, n_r=8)

# create a mesh with a maximum area of 2
mesh = geometry.create_mesh(mesh_sizes=[2])

# create a CrossSection object
section = CrossSection(geometry, mesh)

# perform a geometry and warping analysis
section.calculate_geometric_properties()
section.calculate_warping_properties()
```

(continues on next page)

(continued from previous page)

```
# perform a stress analysis with Mx = 5 kN.m; Vx = 10 kN and Mzz = 3 kN.m
case1 = section.calculate_stress(Mxx=5e6, Vx=10e3, Mzz=3e6)

# perform a stress analysis with My = 15 kN.m; Vy = 30 kN and Mzz = 1.5 kN.m
case2 = section.calculate_stress(Myy=15e6, Vy=30e3, Mzz=1.5e6)

case1.plot_stress_m_zz(pause=False)  # plot the bending stress for case1
case1.plot_vector_mzz_zxy(pause=False)  # plot the torsion vectors for case1
case2.plot_stress_v_zxy(pause=False)  # plot the shear stress for case1
case1.plot_stress_vm(pause=False)  # plot the von mises stress for case1
case2.plot_stress_vm()  # plot the von mises stress for case2
```

The following plots are generated by the above example:

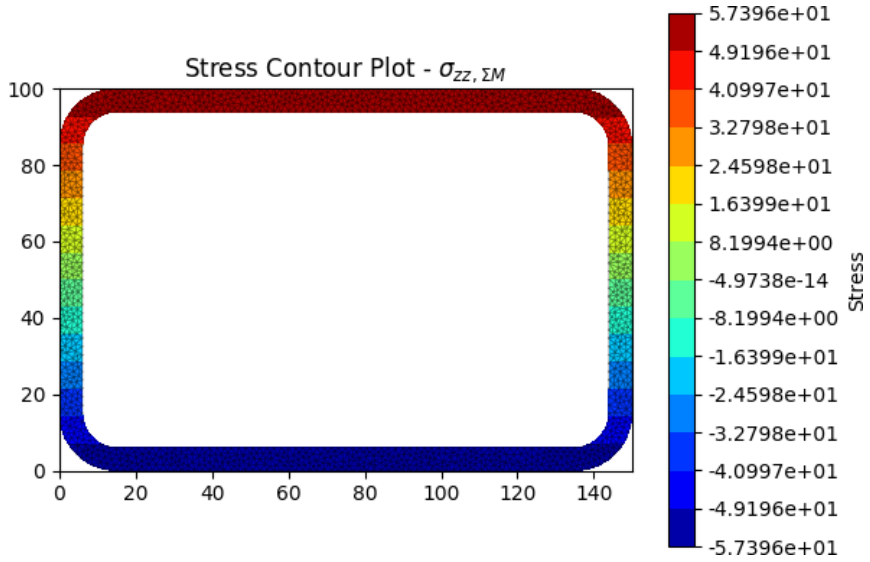


Fig. 14: Contour plot of the bending stress for case 1.

6.7 Creating a Composite Cross-Section

The following example demonstrates how to create a composite cross-section by assigning different material properties to various regions of the mesh. A steel 310UB40.4 is modelled with a 50Dx600W timber panel placed on its top flange.

The geometry and mesh are plotted, and the mesh information printed to the terminal before the analysis is carried out. All types of cross-section analyses are carried out, with an axial force, bending moment and shear force applied during the stress analysis. Once the analysis is complete, the cross-section properties are printed to the terminal and a plot of the centroids and cross-section stresses generated:

```
import sectionproperties.pre.sections as sections
from sectionproperties.pre.pre import Material
from sectionproperties.analysis.cross_section import CrossSection

# create material properties
steel = Material(name='Steel', elastic_modulus=200e3, poissons_ratio=0.3,
                 yield_strength=500, color='grey')
```

(continues on next page)

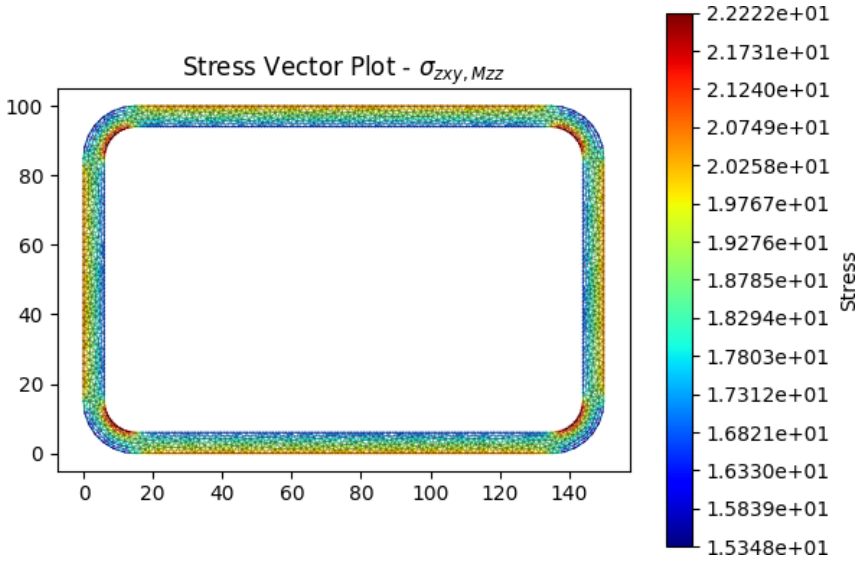


Fig. 15: Vector plot of the torsion stress for case 1.

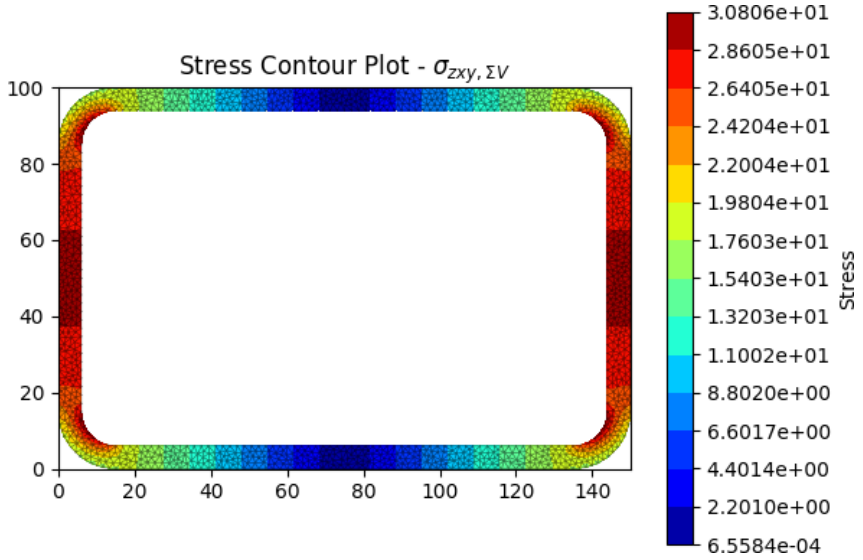


Fig. 16: Contour plot of the shear stress for case 2.

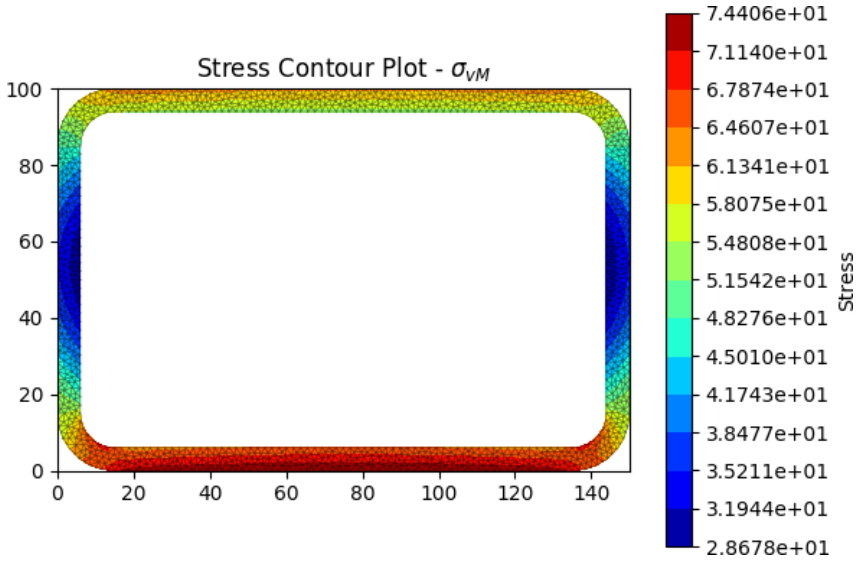


Fig. 17: Contour plot of the von Mises stress for case 1.

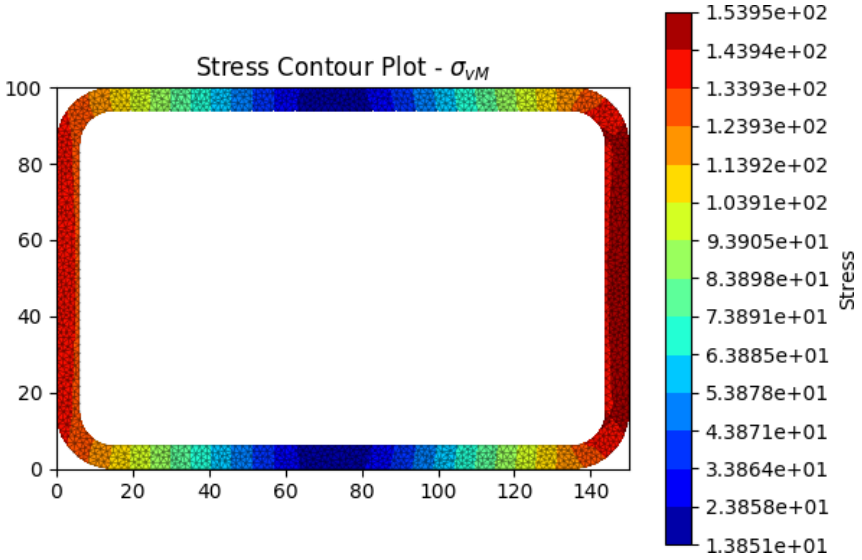


Fig. 18: Contour plot of the von Mises stress for case 2.

(continued from previous page)

```
timber = Material(name='Timber', elastic_modulus=8e3, poissons_ratio=0.35,
                  yield_strength=20, color='burlywood')

# create 310UB40.4
ub = sections.ISection(d=304, b=165, t_f=10.2, t_w=6.1, r=11.4, n_r=8)

# create timber panel on top of the UB
panel = sections.RectangularSection(d=50, b=600, shift=[-217.5, 304])

# merge the two sections into one geometry object
geometry = sections.MergedSection([ub, panel])
geometry.clean_geometry()  # clean the geometry
geometry.plot_geometry()  # plot the geometry

# create a mesh - use a mesh size of 5 for the UB, 20 for the panel
mesh = geometry.create_mesh(mesh_sizes=[5, 20])

# create a CrossSection object - take care to list the materials in the same
# order as entered into the MergedSection
section = CrossSection(geometry, mesh, materials=[steel, timber])
section.display_mesh_info()  # display the mesh information

# plot the mesh with coloured materials and a line transparency of 0.5
section.plot_mesh(materials=True, alpha=0.5)

# perform a geometric, warping and plastic analysis
section.calculate_geometric_properties(time_info=True)
section.calculate_warping_properties(time_info=True)
section.calculate_plastic_properties(time_info=True, verbose=True)

# perform a stress analysis with N = 100 kN, Mxx = 120 kN.m and Vy = 75 kN
stress_post = section.calculate_stress(N=-100e3, Mxx=-120e6, Vy=-75e3,
                                       time_info=True)

# print the results to the terminal
section.display_results()

# plot the centroids
section.plot_centroids()

stress_post.plot_stress_n_zz(pause=False)  # plot the axial stress
stress_post.plot_stress_m_zz(pause=False)  # plot the bending stress
stress_post.plot_stress_v_zxy()  # plot the shear stress
```

The following plots are generated by the above example:

The following is printed to the terminal:

```
Mesh Statistics:
--8972 nodes
--4189 elements
--2 regions

--Calculating geometric section properties...
----completed in 2.619151 seconds---

--Assembing 8972x8972 stiffness matrix and load vector...
```

(continues on next page)

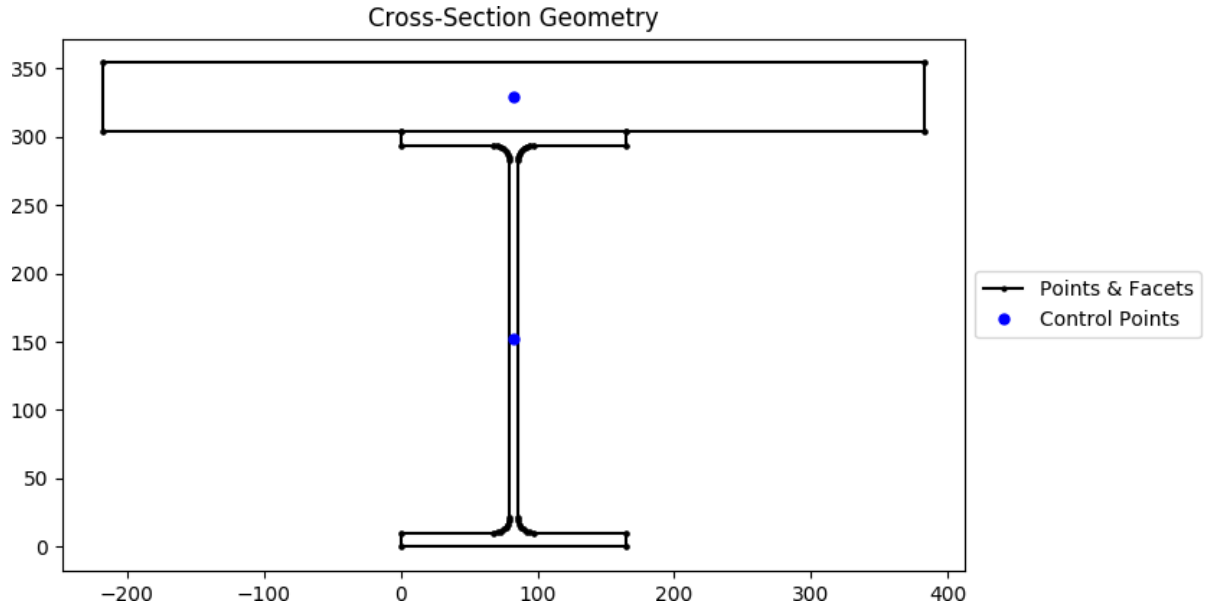


Fig. 19: Plot of the generated geometry object.

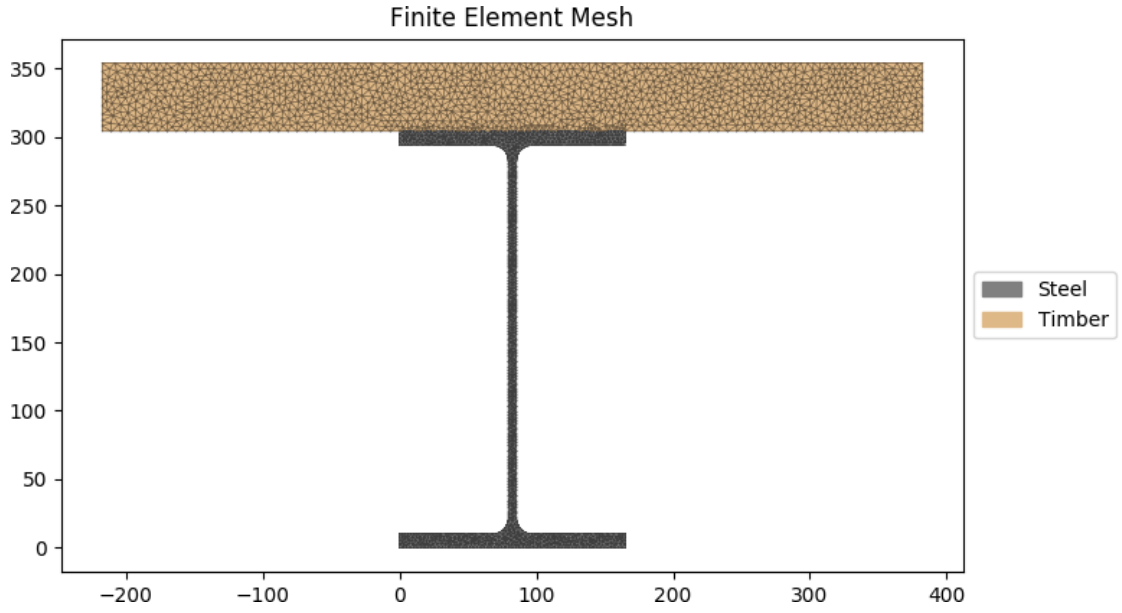


Fig. 20: Mesh generated from the above geometry.

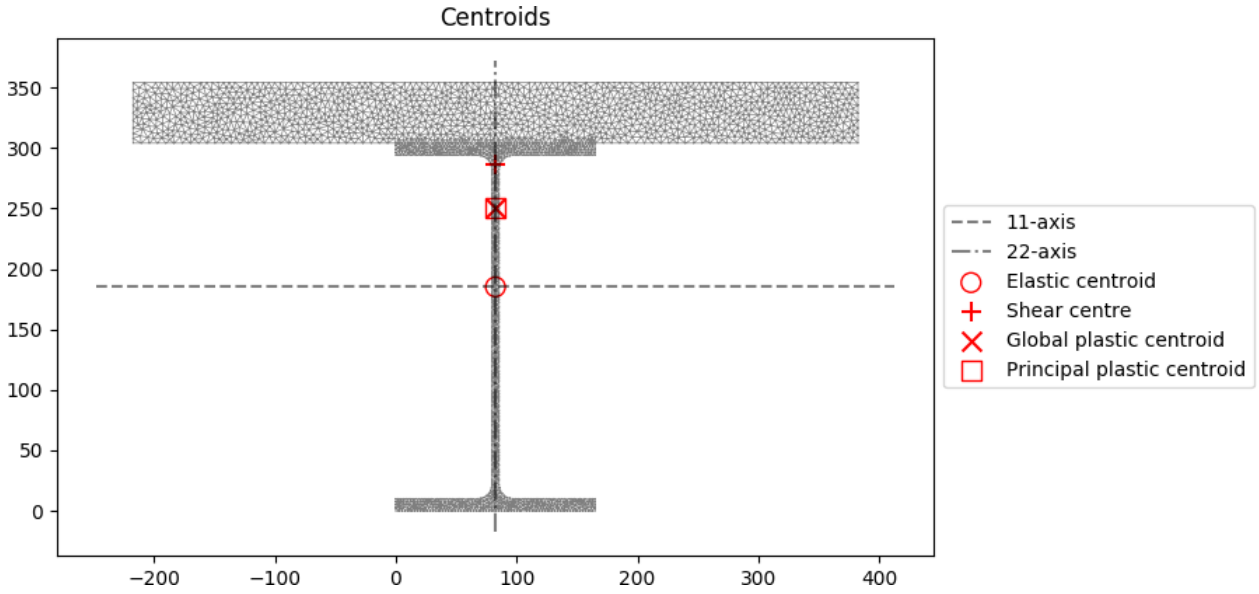


Fig. 21: Plot of the centroids and the principal axis.

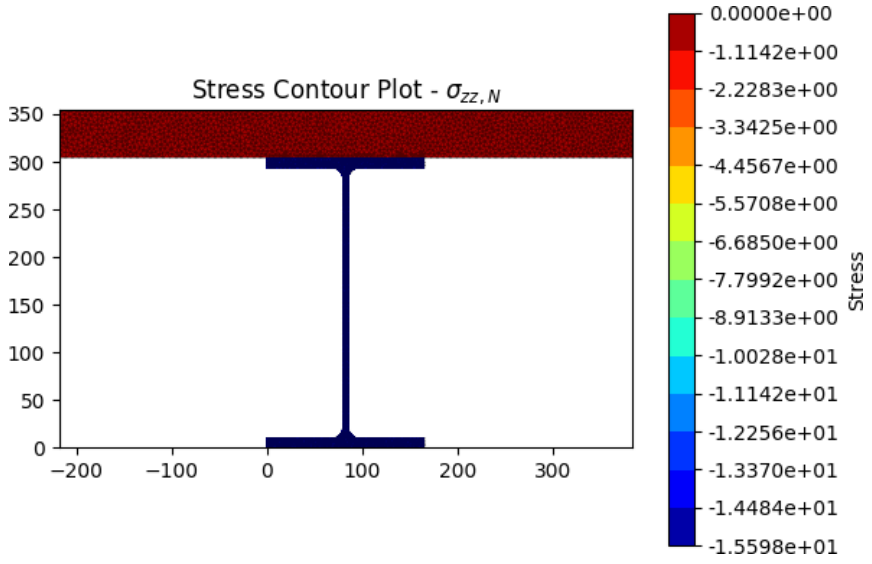


Fig. 22: Contour plot of the axial stress.

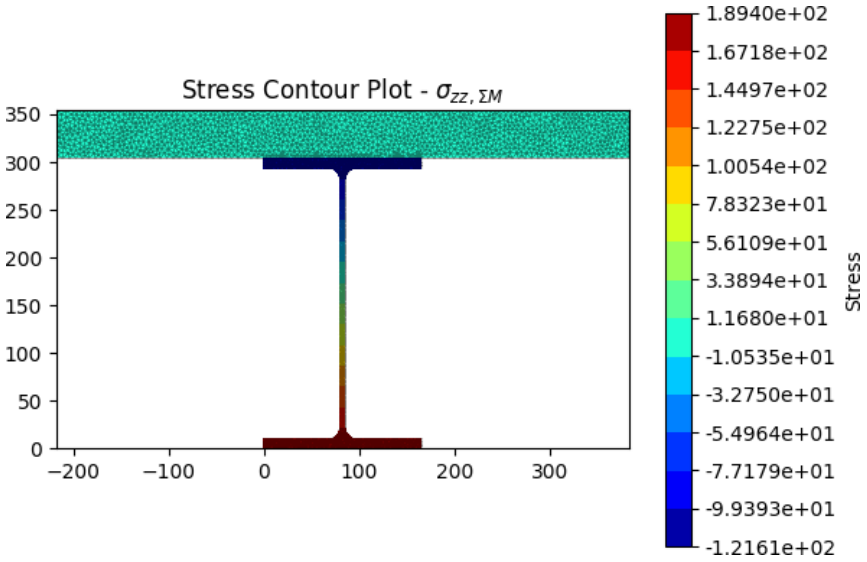


Fig. 23: Contour plot of the bending stress.

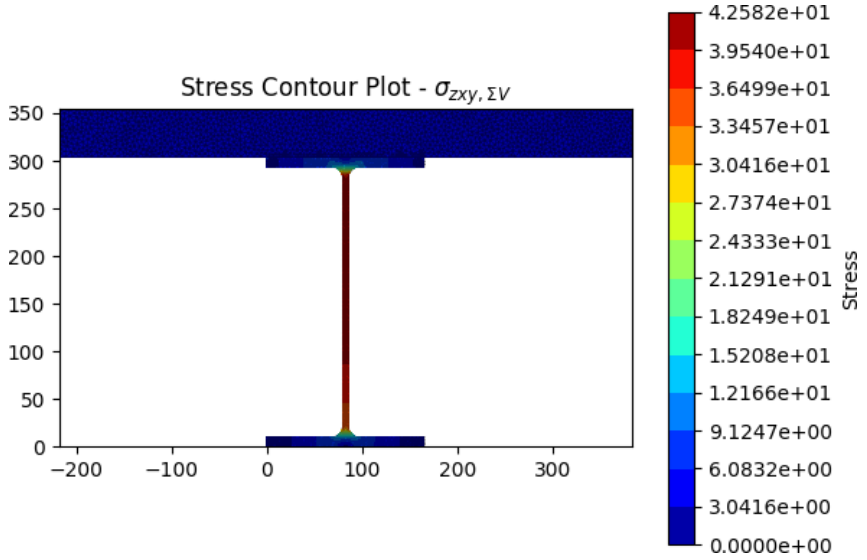


Fig. 24: Contour plot of the shear stress.

(continued from previous page)

```
----completed in 4.814592 seconds---
--Solving for the warping function using the direct solver...
----completed in 0.032710 seconds---
--Computing the torsion constant...
----completed in 0.000281 seconds---
--Assembling shear function load vectors...
----completed in 3.648590 seconds---
--Solving for the shear functions using the direct solver...
----completed in 0.073731 seconds---
--Assembling shear centre and warping moment integrals...
----completed in 2.288843 seconds---
--Calculating shear centres...
----completed in 0.000064 seconds---
--Assembling shear deformation coefficients...
----completed in 3.597728 seconds---
--Assembling monosymmetry integrals...
----completed in 2.519333 seconds---

--Calculating plastic properties...
d = -185.13088495027134; f_norm = 1.0
d = 168.86911504972866; f_norm = -1.0
d = -8.130884950271337; f_norm = 0.1396051884674814
d = 13.552166872240885; f_norm = 0.0983423820518053
d = 60.60270845168385; f_norm = 0.008805290832496546
d = 64.90008929872263; f_norm = 0.0006273832465500235
d = 65.22746216923525; f_norm = 4.393296044849056e-06
d = 65.22976962543858; f_norm = 2.2112746988930985e-09
d = 65.2298027403234; f_norm = -6.080628821651535e-08
---x-axis plastic centroid calculation converged at 6.52298e+01 in 8 iterations.
d = -300.0; f_norm = -1.0
d = 300.0; f_norm = 1.0
d = 0.0; f_norm = 2.1790636349700628e-16
d = -5e-07; f_norm = -4.7730935851751974e-08
---y-axis plastic centroid calculation converged at 0.00000e+00 in 3 iterations.
d = -185.13088495027134; f_norm = 1.0
d = 168.86911504972866; f_norm = -1.0
d = -8.130884950271337; f_norm = 0.1396051884674814
d = 13.552166872240885; f_norm = 0.0983423820518053
d = 60.60270845168385; f_norm = 0.008805290832496546
d = 64.90008929872263; f_norm = 0.0006273832465500235
d = 65.22746216923525; f_norm = 4.393296044849056e-06
d = 65.22976962543858; f_norm = 2.2112746988930985e-09
d = 65.2298027403234; f_norm = -6.080628821651535e-08
---11-axis plastic centroid calculation converged at 6.52298e+01 in 8 iterations.
d = -300.0; f_norm = -1.0
d = 300.0; f_norm = 1.0
d = 0.0; f_norm = 2.1790636349700628e-16
d = -5e-07; f_norm = -4.7730935851751974e-08
---22-axis plastic centroid calculation converged at 0.00000e+00 in 3 iterations.
----completed in 0.794056 seconds---

--Calculating cross-section stresses...
----completed in 4.240446 seconds---

Section Properties:
A      = 3.521094e+04
E.A    = 1.282187e+09
```

(continues on next page)

(continued from previous page)

```
E.Qx   = 2.373725e+11
E.Qy   = 1.057805e+11
cx     = 8.250000e+01
cy     = 1.851309e+02
E.Ixx_g= 6.740447e+13
E.Iyy_g= 1.745613e+13
E.Ixy_g= 1.958323e+13
E.Ixx_c= 2.345949e+13
E.Iyy_c= 8.729240e+12
E.Ixy_c= -7.421875e-02
E.Zxx+ = 1.389212e+11
E.Zxx- = 1.267184e+11
E.Zyy+ = 2.909747e+10
E.Zyy- = 2.909747e+10
rx     = 1.352644e+02
ry     = 8.251112e+01
phi    = 0.000000e+00
E.I11_c= 2.345949e+13
E.I22_c= 8.729240e+12
E.Z11+ = 1.389212e+11
E.Z11- = 1.267184e+11
E.Z22+ = 2.909747e+10
E.Z22- = 2.909747e+10
r11    = 1.352644e+02
r22    = 8.251112e+01
G.J    = 1.439379e+11
G.Iw   = 2.554353e+16
x_se   = 8.250071e+01
y_se   = 2.863400e+02
x_st   = 8.250070e+01
y_st   = 2.857074e+02
x1_se  = 7.063407e-04
y2_se  = 1.012091e+02
A_sx   = 1.104723e+04
A_sy   = 1.021183e+04
A_s11  = 1.104723e+04
A_s22  = 1.021183e+04
betax+ = 2.039413e+02
betax- = -2.039413e+02
betay+ = 1.412681e-03
betay- = -1.412681e-03
beta11+= 2.039413e+02
beta11-= -2.039413e+02
beta22+= 1.412681e-03
beta22-= -1.412681e-03
x_pc   = 8.250000e+01
y_pc   = 2.503607e+02
M_p,xx = 3.932542e+08
M_p,yy = 1.610673e+08
x11_pc = 8.250000e+01
y22_pc = 2.503607e+02
M_p,11 = 3.932542e+08
M_p,22 = 1.610673e+08
```

6.8 Frame Analysis Example

The following example demonstrates how *sectionproperties* can be used to calculate the cross-section properties required for a frame analysis. Using this method is preferred over executing a geometric and warping analysis as only variables required for a frame analysis are computed. In this example the torsion constant of a rectangular section is calculated for a number of different mesh sizes and the accuracy of the result compared with the time taken to obtain the solution:

```
import time
import numpy as np
import matplotlib.pyplot as plt
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

# create a rectangular section
geometry = sections.RectangularSection(d=100, b=50)

# create a list of mesh sizes to analyse
mesh_sizes = [1.5, 2, 2.5, 3, 4, 5, 10, 15, 20, 25, 30, 40, 50, 75, 100]
j_calc = []  # list to store torsion constants
t_calc = []  # list to store computation times

# loop through mesh sizes
for mesh_size in mesh_sizes:
    mesh = geometry.create_mesh(mesh_sizes=[mesh_size])  # create mesh
    section = CrossSection(geometry, mesh)  # create a CrossSection object
    start_time = time.time()  # start timing
    # calculate the frame properties
    (_, _, _, _, j, _) = section.calculate_frame_properties()
    t = time.time() - start_time  # stop timing
    t_calc.append(t)  # save the time
    j_calc.append(j)  # save the torsion constant
    # print the result
    str = "Mesh Size: {0}; ".format(mesh_size)
    str += "Solution Time {0:.5f} s; ".format(t)
    str += "Torsion Constant: {0:.12e}".format(j)
    print(str)

correct_val = j_calc[0]  # assume the finest mesh gives the 'correct' value
j_np = np.array(j_calc)  # convert results to a numpy array
error_vals = (j_calc - correct_val) / j_calc * 100  # compute the error

# produce a plot of the accuracy of the torsion constant with computation time
plt.loglog(t_calc[1:], error_vals[1:], 'kx-')
plt.xlabel("Solver Time [s]")
plt.ylabel("Torsion Constant Error [%]")
plt.show()
```

6.9 Advanced Examples

The following examples demonstrates how *sectionproperties* can be used for more academic purposes.

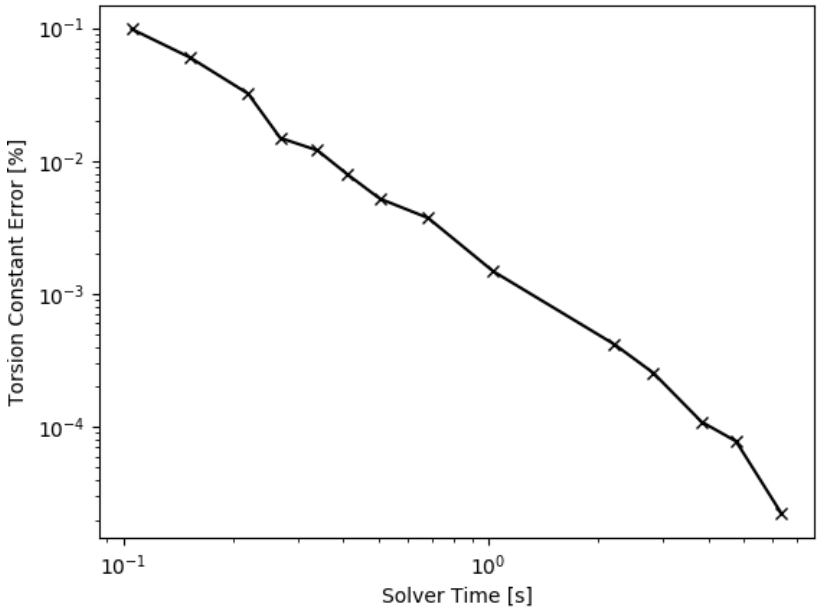


Fig. 25: Plot of the torsion constant as a function of the solution time.

6.9.1 Torsion Constant of a Rectangle

In this example, the aspect ratio of a rectangular section is varied whilst keeping a constant cross-sectional area and the torsion constant calculated. The variation of the torsion constant with the aspect ratio is then plotted:

```
import numpy as np
import matplotlib.pyplot as plt
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

# rectangle dimensions
d_list = []
b_list = np.linspace(0.2, 1, 20)
j_list = []  # list holding torsion constant results

# number of elements for each analysis
n = 500

# loop through all the widths
for b in b_list:
    # calculate d assuming area = 1
    d = 1 / b
    d_list.append(d)

    # compute mesh size
    ms = d * b / n

    # perform a warping analysis on rectangle
    geometry = sections.RectangularSection(d=d, b=b)
    mesh = geometry.create_mesh(mesh_sizes=[ms])
    section = CrossSection(geometry, mesh)
    section.calculate_geometric_properties()
    section.calculate_warping_properties()
```

(continues on next page)

(continued from previous page)

```
    # get the torsion constant
    j = section.get_j()
    print("d/b = {0:.3f}; J = {1:.5e}".format(d/b, j))
    j_list.append(j)

# plot the torsion constant as a function of the aspect ratio
(fig, ax) = plt.subplots()
ax.plot(np.array(d_list) / b_list, j_list, 'kx-')
ax.set_xlabel("Aspect Ratio [d/b]")
ax.set_ylabel("Torsion Constant [J]")
ax.set_title("Rectangular Section Torsion Constant")
plt.show()
```

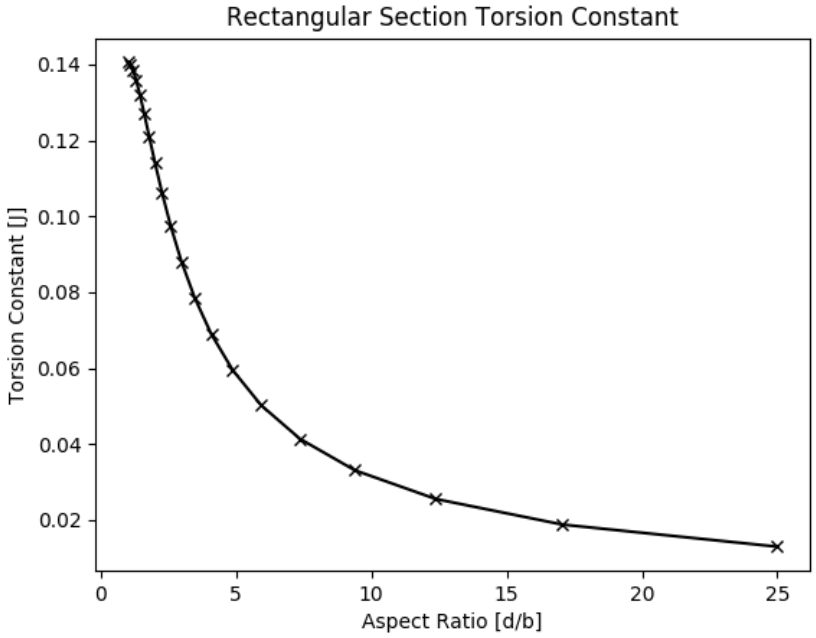


Fig. 26: Plot of the torsion constant as a function of the aspect ratio.

6.9.2 Mesh Refinement

In this example the convergence of the torsion constant is investigated through an analysis of an I-section. The mesh is refined both by modifying the mesh size and by specifying the number of points making up the root radius. The figure below the example code shows that mesh refinement adjacent to the root radius is a far more efficient method in obtaining fast convergence when compared to reducing the mesh area size for the entire section:

```
import numpy as np
import matplotlib.pyplot as plt
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

# define mesh sizes
```

(continues on next page)

(continued from previous page)

```
mesh_size_list = [50, 20, 10, 5, 3, 2, 1]
nr_list = [4, 8, 12, 16, 20, 24, 32, 64]

# initialise result lists
mesh_results = []
mesh_elements = []
nr_results = []
nr_elements = []

# calculate reference solution
geometry = sections.ISection(d=203, b=133, t_f=7.8, t_w=5.8, r=8.9, n_r=64)
mesh = geometry.create_mesh(mesh_sizes=[0.5])  # create mesh
section = CrossSection(geometry, mesh)  # create a CrossSection object
section.calculate_geometric_properties()
section.calculate_warping_properties()
j_reference = section.get_j()  # get the torsion constant

# run through mesh_sizes with n_r = 16
for mesh_size in mesh_size_list:
    geometry = sections.ISection(d=203, b=133, t_f=7.8, t_w=5.8, r=8.9, n_r=16)
    mesh = geometry.create_mesh(mesh_sizes=[mesh_size])  # create mesh
    section = CrossSection(geometry, mesh)  # create a CrossSection object
    section.calculate_geometric_properties()
    section.calculate_warping_properties()

    mesh_elements.append(len(section.elements))
    mesh_results.append(section.get_j())

# run through n_r with mesh_size = 3
for n_r in nr_list:
    geometry = sections.ISection(d=203, b=133, t_f=7.8, t_w=5.8, r=8.9, n_r=n_r)
    mesh = geometry.create_mesh(mesh_sizes=[3])  # create mesh
    section = CrossSection(geometry, mesh)  # create a CrossSection object
    section.calculate_geometric_properties()
    section.calculate_warping_properties()

    nr_elements.append(len(section.elements))
    nr_results.append(section.get_j())

# convert results to a numpy array
mesh_results = np.array(mesh_results)
nr_results = np.array(nr_results)

# compute the error
mesh_error_vals = (mesh_results - j_reference) / mesh_results * 100
nr_error_vals = (nr_results - j_reference) / nr_results * 100

# plot the results
(fig, ax) = plt.subplots()
ax.loglog(mesh_elements, mesh_error_vals, 'kx-', label='Mesh Size Refinement')
ax.loglog(nr_elements, nr_error_vals, 'rx-', label='Root Radius Refinement')
plt.xlabel("Number of Elements")
plt.ylabel("Torsion Constant Error [%]")
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()
```

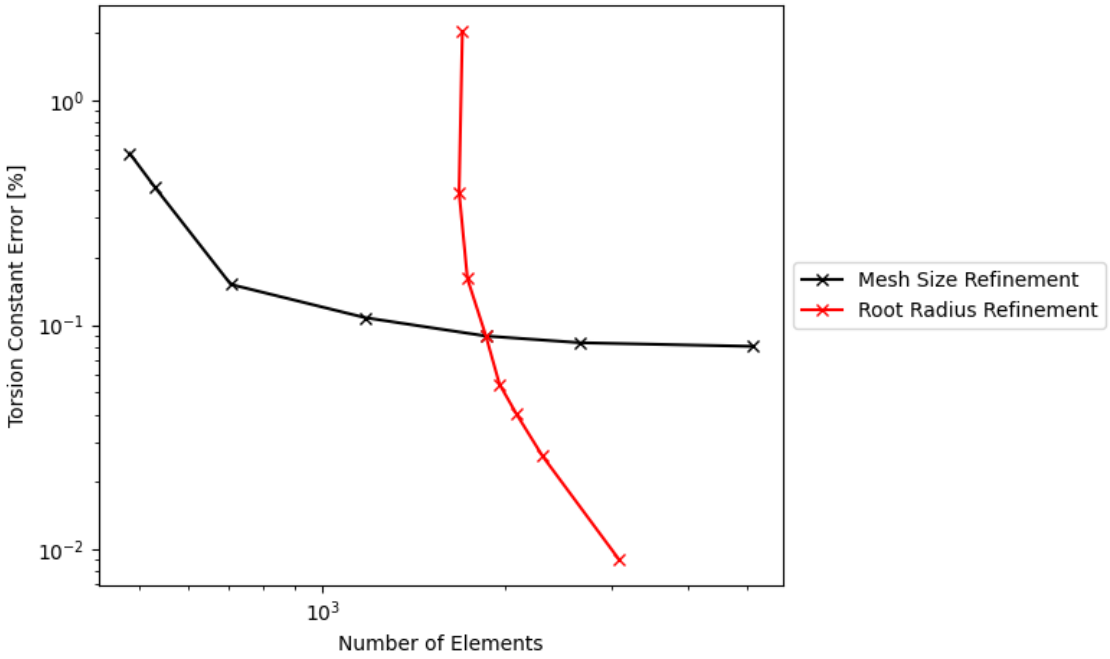


Fig. 27: Plot of the torsion constant error as a function of number of elements used in the analysis for both general mesh refinement and root radius refinement.

CHAPTER 7

Python API Documentation

7.1 Pre-Processor Package

7.1.1 *sections* Module

Geometry Class

class sectionproperties.pre.sections.**Geometry**(*control_points*, *shift*)

Parent class for a cross-section geometry input.

Provides an interface for the user to specify the geometry defining a cross-section. A method is provided for generating a triangular mesh, for translating the cross-section by *(x, y)* and for plotting the geometry.

Variables

- **points** (*list[list[float, float]]*) – List of points *(x, y)* defining the vertices of the cross-section
- **facets** (*list[list[int, int]]*) – List of point index pairs *(p1, p2)* defining the edges of the cross-section
- **holes** (*list[list[float, float]]*) – List of points *(x, y)* defining the locations of holes within the cross-section. If there are no holes, provide an empty list [].
- **control_points** (*list[list[float, float]]*) – A list of points *(x, y)* that define different regions of the cross-section. A control point is an arbitrary point within a region enclosed by facets.
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*
- **perimeter** (*list[int]*) – List of facet indices defining the perimeter of the cross-section

add_control_point(*control_point*)

Adds a control point to the geometry and returns the added control point id.

Parameters **hole** (*list[float, float]*) – Location of the control point

Returns Control point id

Return type int

add_facet(*facet*)
Adds a facet to the geometry and returns the added facet id.

Parameters **facet** (*list[float, float]*) – Point indices of the facet

Returns Facet id

Return type int

add_hole(*hole*)
Adds a hole location to the geometry and returns the added hole id.

Parameters **hole** (*list[float, float]*) – Location of the hole

Returns Hole id

Return type int

add_point(*point*)
Adds a point to the geometry and returns the added point id.

Parameters **point** (*list[float, float]*) – Location of the point

Returns Point id

Return type int

calculate_extents()
Calculates the minimum and maximum x and y-values amongst the list of points.

Returns Minimum and maximum x and y-values *(x_min, x_max, y_min, y_max)*

Return type tuple(float, float, float, float)

calculate_facet_length(*facet*)
Calculates the length of the facet.

Parameters **facet** – Point index pair *(p1, p2)* defining a facet

Returns Facet length

Return type float

calculate_perimeter()
Calculates the perimeter of the cross-section by summing the length of all facets in the `perimeter` class variable.

Returns Cross-section perimeter, returns 0 if there is no perimeter defined

Return type float

clean_geometry(*verbose=False*)
Peforms a full clean on the geometry.

Parameters **verbose** (*bool*) – If set to true, information related to the geometry cleaning process is printed to the terminal.

Note: Cleaning the geometry is always recommended when creating a merged section, which may result in overlapping or intersecting facets, or duplicate nodes.

---

create_mesh(*mesh_sizes*)

Creates a quadratic triangular mesh from the Geometry object.

Parameters **mesh_sizes** – A list of maximum element areas corresponding to each region within the cross-section geometry.

Returns Object containing generated mesh data

Return type `meshpy.triangle.MeshInfo`

Raises **AssertionError** – If the number of mesh sizes does not match the number of regions

The following example creates a circular cross-section with a diameter of 50 with 64 points, and generates a mesh with a maximum triangular area of 2.5:

```
import sectionproperties.pre.sections as sections

geometry = sections.CircularSection(d=50, n=64)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
```

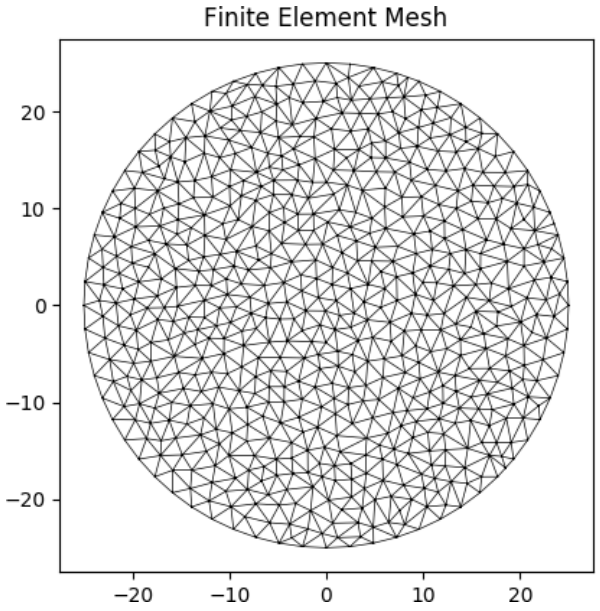


Fig. 1: Mesh generated from the above geometry.

draw_radius(*pt*, *r*, *theta*, *n*, *anti=True*)

Adds a quarter radius of points to the points list - centered at point *pt*, with radius *r*, starting at angle *theta*, with *n* points. If r = 0, adds pt only.

Parameters

- **pt** (*list[float, float]*) – Centre of radius *(x,y)*
- **r** (*float*) – Radius
- **theta** (*float*) – Initial angle
- **n** (*int*) – Number of points
- **anti** (*bool*) – Anticlockwise rotation?

mirror_section(*axis='x'*, *mirror_point=None*)

Mirrors the geometry about a point on either the x or y-axis. If no point is provided, mirrors the geometry about the first control point in the list of control points of the *Geometry* object.

Parameters

- **axis** (*string*) – Axis about which to mirror the geometry, *'x'* or *'y'*
- **mirror_point** (*list[float, float]*) – Point about which to mirror the geometry *(x, y)*

The following example mirrors a 200PFC section about the y-axis and the point (0, 0):

```
import sectionproperties.pre.sections as sections

geometry = sections.PfcSection(d=200, b=75, t_f=12, t_w=6, r=12, n_r=8)
geometry.mirror_section(axis='y', mirror_point=[0, 0])
```

plot_geometry(*ax=None*, *pause=True*, *labels=False*, *perimeter=False*)

Plots the geometry defined by the input section. If no axes object is supplied a new figure and axis is created.

Parameters

- **ax** (matplotlib.axes.Axes) – Axes object on which the mesh is plotted
- **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.
- **labels** (*bool*) – If set to true, node and facet labels are displayed
- **perimeter** (*bool*) – If set to true, boldens the perimeter of the cross-section

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example creates a CHS discretised with 64 points, with a diameter of 48 and thickness of 3.2, and plots the geometry:

```
import sectionproperties.pre.sections as sections

geometry = sections.Chs(d=48, t=3.2, n=64)
geometry.plot_geometry()
```

rotate_section(*angle*, *rot_point=None*)

Rotates the geometry and specified angle about a point. If the rotation point is not provided, rotates the section about the first control point in the list of control points of the *Geometry* object.

Parameters

- **angle** (*float*) – Angle (degrees) by which to rotate the section. A positive angle leads to a counter-clockwise rotation.
- **rot_point** (*list[float, float]*) – Point *(x, y)* about which to rotate the section

The following example rotates a 200UB25 section clockwise by 30 degrees:

```
import sectionproperties.pre.sections as sections

geometry = sections.ISection(d=203, b=133, t_f=7.8, t_w=5.8, r=8.9, n_r=8)
geometry.rotate_section(angle=-30)
```

shift_section()

Shifts the cross-section parameters by the class variable vector *shift*.

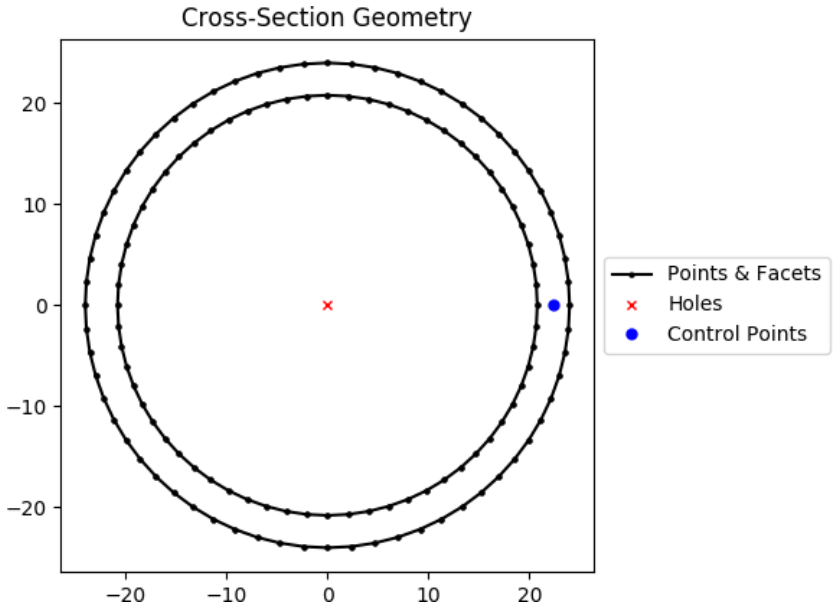


Fig. 2: Geometry generated by the above example.

CustomSection Class

class sectionproperties.pre.sections.**CustomSection**(*points, facets, holes, control_points, shift=[0, 0], perimeter=[]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a cross-section from a list of points, facets, holes and a user specified control point.

Parameters

- **points** (*list[list[float, float]]*) – List of points *(x, y)* defining the vertices of the cross-section
- **facets** (*list[list[int, int]]*) – List of point index pairs *(p1, p2)* defining the edges of the cross-section
- **holes** (*list[list[float, float]]*) – List of points *(x, y)* defining the locations of holes within the cross-section. If there are no holes, provide an empty list [].
- **control_points** (*list[list[float, float]]*) – A list of points *(x, y)* that define different regions of the cross-section. A control point is an arbitrary point within a region enclosed by facets.
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*
- **perimeter** – List of facet indices defining the perimeter of the cross-section

The following example creates a hollow trapezium with a base width of 100, top width of 50, height of 50 and a wall thickness of 10. A mesh is generated with a maximum triangular area of 2.0:

```
import sectionproperties.pre.sections as sections

points = [[0, 0], [100, 0], [75, 50], [25, 50], [15, 10], [85, 10], [70, 40], [30,
↪ 40]]
facets = [[0, 1], [1, 2], [2, 3], [3, 0], [4, 5], [5, 6], [6, 7], [7, 4]]
```

(continues on next page)

(continued from previous page)

```
holes = [[50, 25]]
control_points = [[5, 5]]
perimeter = [0, 1, 2, 3]

geometry = sections.CustomSection(
    points, facets, holes, control_points, perimeter=perimeter
)
mesh = geometry.create_mesh(mesh_sizes=[2.0])
```

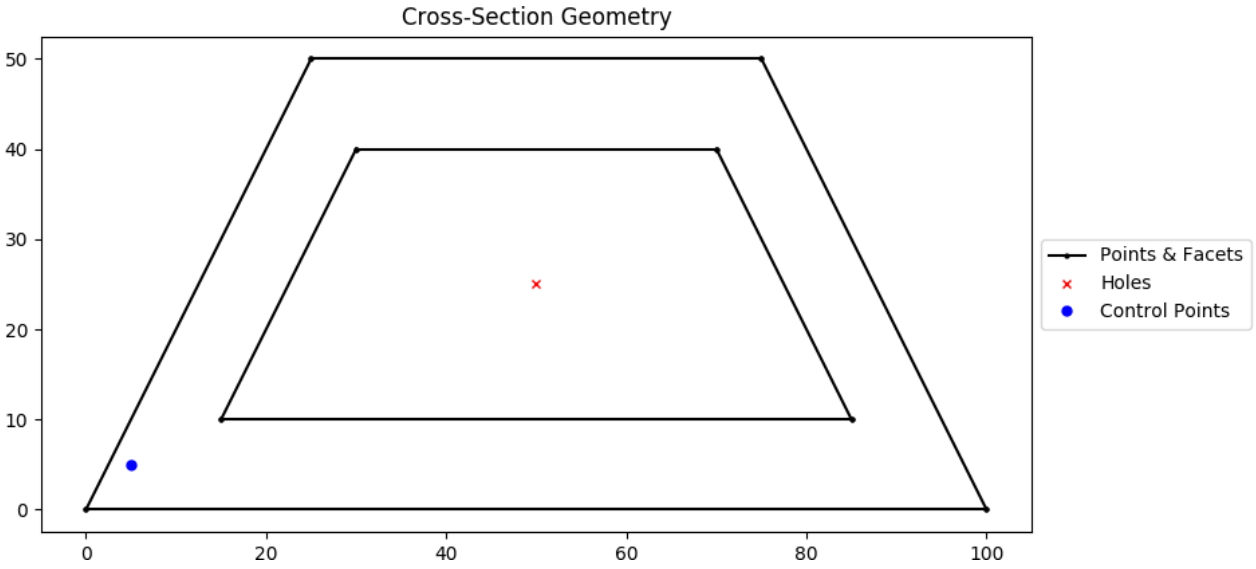


Fig. 3: Custom section geometry.

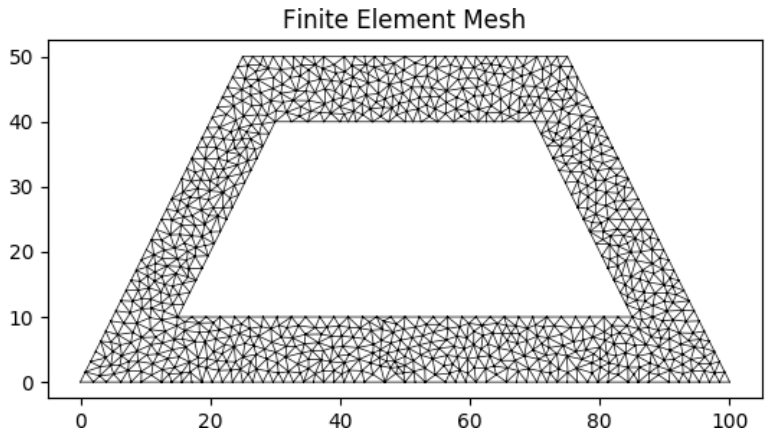


Fig. 4: Mesh generated from the above geometry.

RectangularSection Class

class sectionproperties.pre.sections.**RectangularSection**(*d*, *b*, *shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a rectangular section with the bottom left corner at the origin *(0, 0)*, with depth *d* and width *b*.

Parameters

- **d** (*float*) – Depth (y) of the rectangle
- **b** (*float*) – Width (x) of the rectangle
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a rectangular cross-section with a depth of 100 and width of 50, and generates a mesh with a maximum triangular area of 5:

```
import sectionproperties.pre.sections as sections

geometry = sections.RectangularSection(d=100, b=50)
mesh = geometry.create_mesh(mesh_sizes=[5])
```

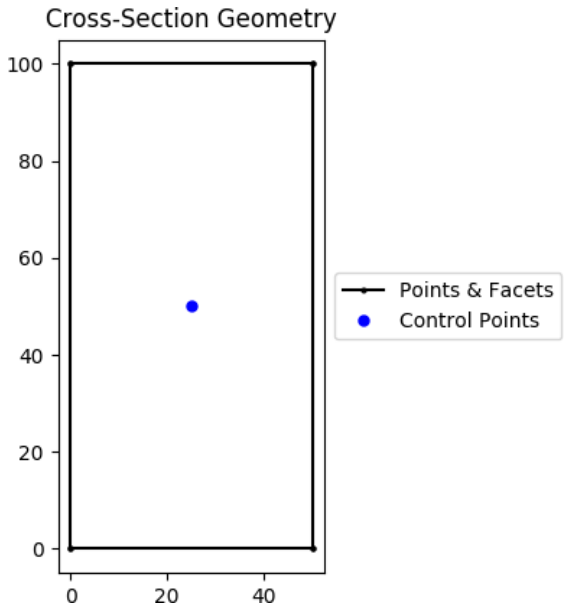


Fig. 5: Rectangular section geometry.

CircularSection Class

class sectionproperties.pre.sections.**CircularSection**(*d, n, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a solid circle centered at the origin *(0, 0)* with diameter *d* and using *n* points to construct the circle.

Parameters

- **d** (*float*) – Diameter of the circle
- **n** (*int*) – Number of points discretising the circle
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a circular cross-section with a diameter of 50 with 64 points, and generates a mesh with a maximum triangular area of 2.5:

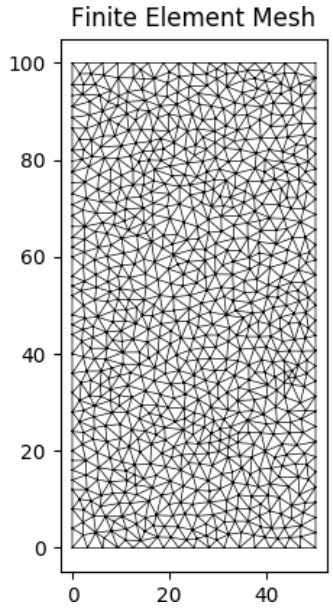


Fig. 6: Mesh generated from the above geometry.

```
import sectionproperties.pre.sections as sections

geometry = sections.CircularSection(d=50, n=64)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
```

Chs Class

class sectionproperties.pre.sections.**Chs**(*d, t, n, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a circular hollow section centered at the origin *(0, 0)*, with diameter *d* and thickness *t*, using *n* points to construct the inner and outer circles.

Parameters

- **d** (*float*) – Outer diameter of the CHS
- **t** (*float*) – Thickness of the CHS
- **n** (*int*) – Number of points discretising the inner and outer circles
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a CHS discretised with 64 points, with a diameter of 48 and thickness of 3.2, and generates a mesh with a maximum triangular area of 1.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.Chs(d=48, t=3.2, n=64)
mesh = geometry.create_mesh(mesh_sizes=[1.0])
```

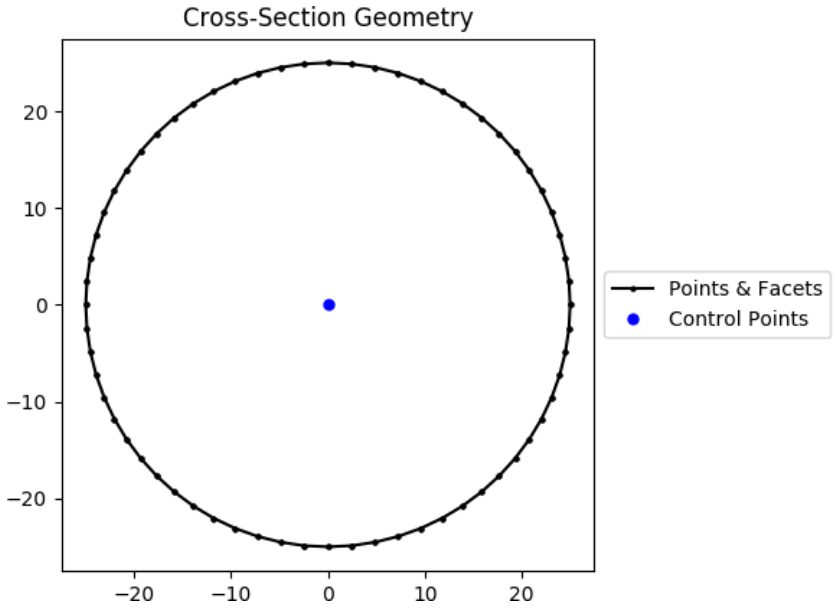


Fig. 7: Circular section geometry.

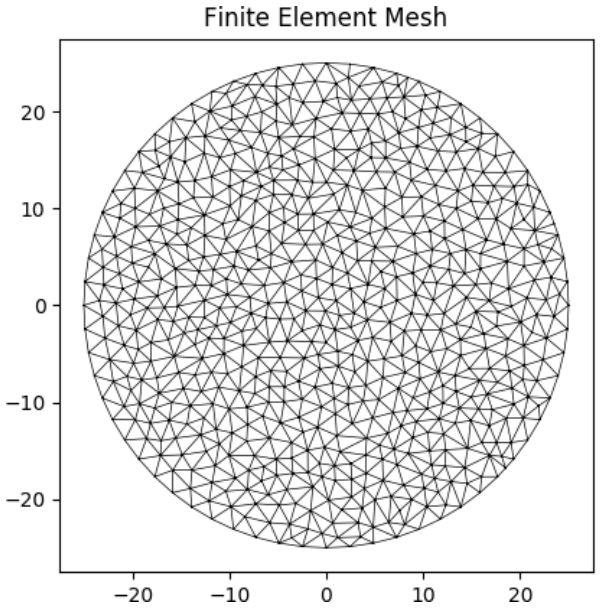


Fig. 8: Mesh generated from the above geometry.

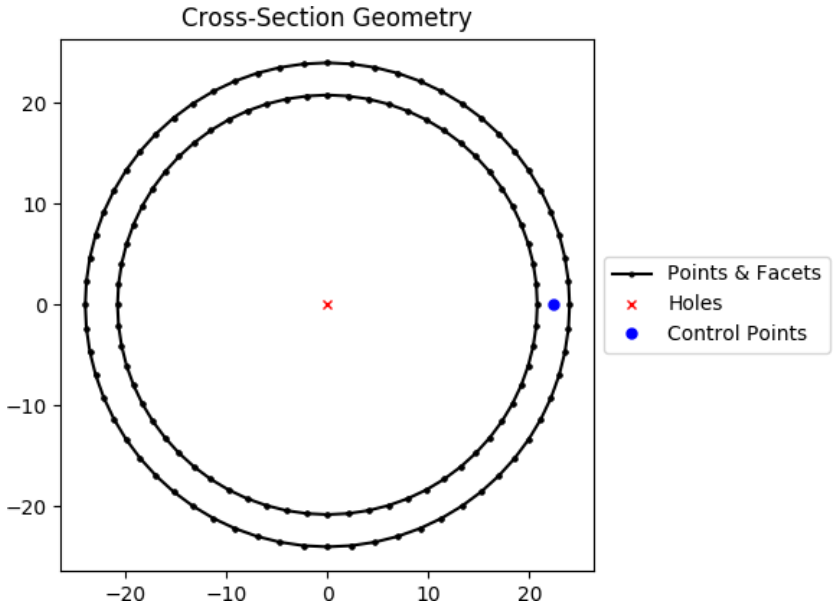


Fig. 9: CHS geometry.

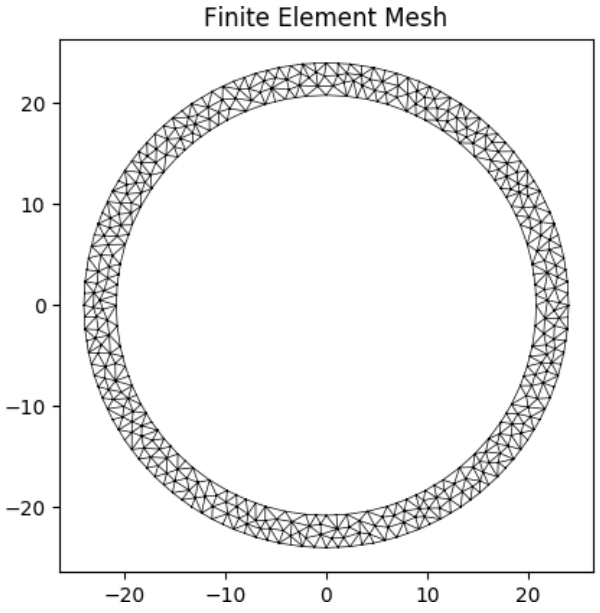


Fig. 10: Mesh generated from the above geometry.

EllipticalSection Class

class sectionproperties.pre.sections.**EllipticalSection**(*d_y, d_x, n, shift=[0, 0]*)
Bases: *sectionproperties.pre.sections.Geometry*

Constructs a solid ellipse centered at the origin *(0, 0)* with vertical diameter *d_y* and horizontal diameter *d_x*, using *n* points to construct the ellipse.

Parameters

- **d_y** (*float*) – Diameter of the ellipse in the y-dimension
- **d_x** (*float*) – Diameter of the ellipse in the x-dimension
- **n** (*int*) – Number of points discretising the ellipse
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates an elliptical cross-section with a vertical diameter of 25 and horizontal diameter of 50, with 40 points, and generates a mesh with a maximum triangular area of 1.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.EllipticalSection(d_y=25, d_x=50, n=40)
mesh = geometry.create_mesh(mesh_sizes=[1.0])
```

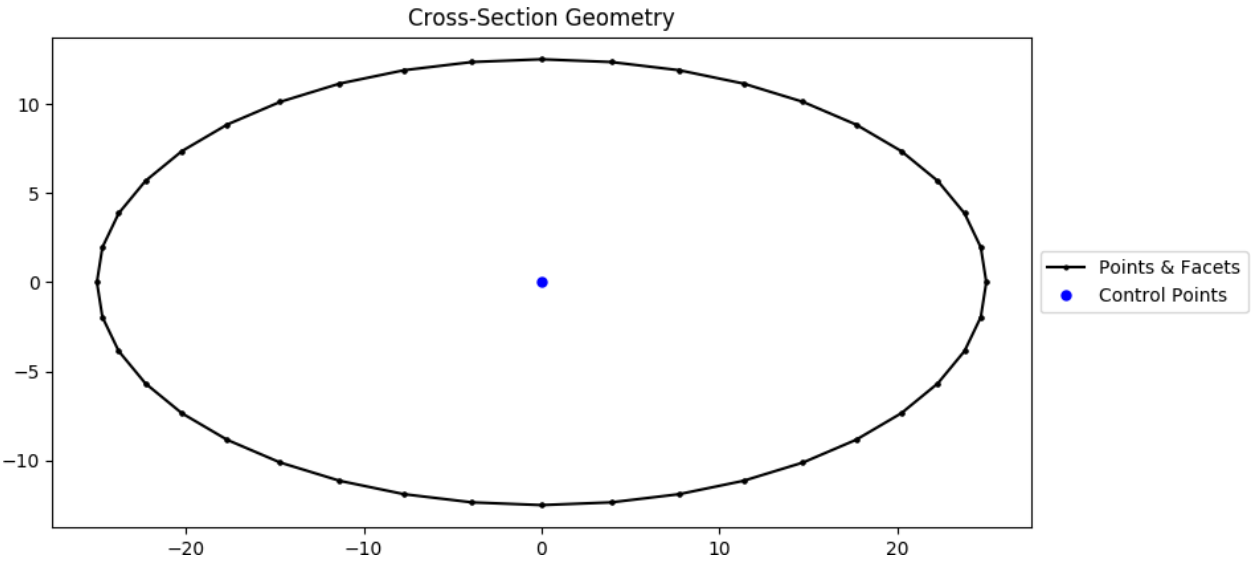


Fig. 11: Elliptical section geometry.

Ehs Class

class sectionproperties.pre.sections.**Ehs**(*d_y, d_x, t, n, shift=[0, 0]*)
Bases: *sectionproperties.pre.sections.Geometry*

Constructs an elliptical hollow section centered at the origin *(0, 0)*, with outer vertical diameter *d_y*, outer horizontal diameter *d_x*, and thickness *t*, using *n* points to construct the inner and outer ellipses.

Parameters

- **d_y** (*float*) – Diameter of the ellipse in the y-dimension

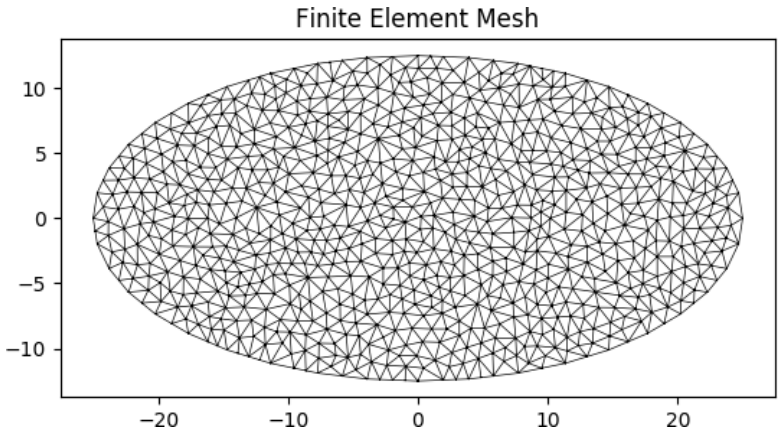


Fig. 12: Mesh generated from the above geometry.

- **d_x** (*float*) – Diameter of the ellipse in the x-dimension
- **t** (*float*) – Thickness of the EHS
- **n** (*int*) – Number of points discretising the inner and outer ellipses
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a EHS discretised with 30 points, with a outer vertical diameter of 25, outer horizontal diameter of 50, and thickness of 2.0, and generates a mesh with a maximum triangular area of 0.5:

```
import sectionproperties.pre.sections as sections

geometry = sections.Ehs(d_y=25, d_x=50, t=2.0, n=64)
mesh = geometry.create_mesh(mesh_sizes=[0.5])
```

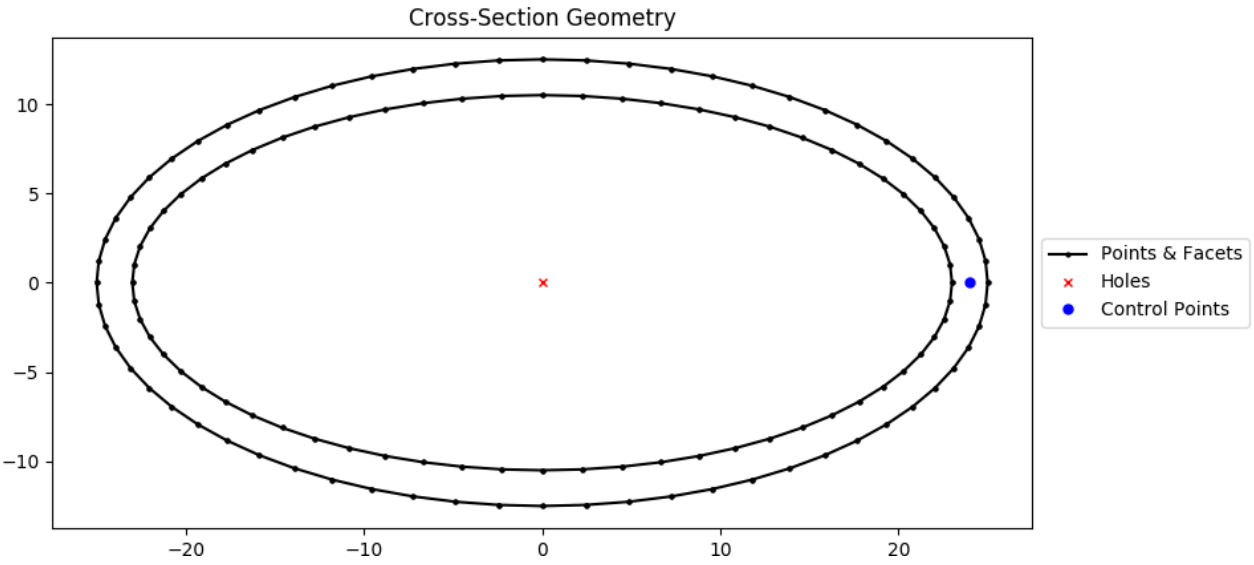


Fig. 13: EHS geometry.

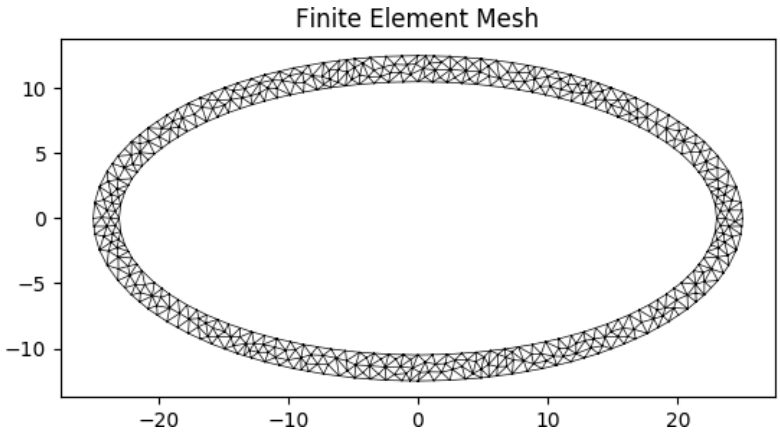


Fig. 14: Mesh generated from the above geometry.

Rhs Class

class sectionproperties.pre.sections.**Rhs**(*d, b, t, r_out, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a rectangular hollow section centered at *(b/2, d/2)*, with depth *d*, width *b*, thickness *t* and outer radius *r_out*, using *n_r* points to construct the inner and outer radii. If the outer radius is less than the thickness of the RHS, the inner radius is set to zero.

Parameters

- **d** (*float*) – Depth of the RHS
- **b** (*float*) – Width of the RHS
- **t** (*float*) – Thickness of the RHS
- **r_out** (*float*) – Outer radius of the RHS
- **n_r** (*int*) – Number of points discretising the inner and outer radii
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates an RHS with a depth of 100, a width of 50, a thickness of 6 and an outer radius of 9, using 8 points to discretise the inner and outer radii. A mesh is generated with a maximum triangular area of 2.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.Rhs(d=100, b=50, t=6, r_out=9, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.0])
```

ISection Class

class sectionproperties.pre.sections.**ISection**(*d, b, t_f, t_w, r, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs an I-section centered at *(b/2, d/2)*, with depth *d*, width *b*, flange thickness *t_f*, web thickness *t_w*, and root radius *r*, using *n_r* points to construct the root radius.

Parameters

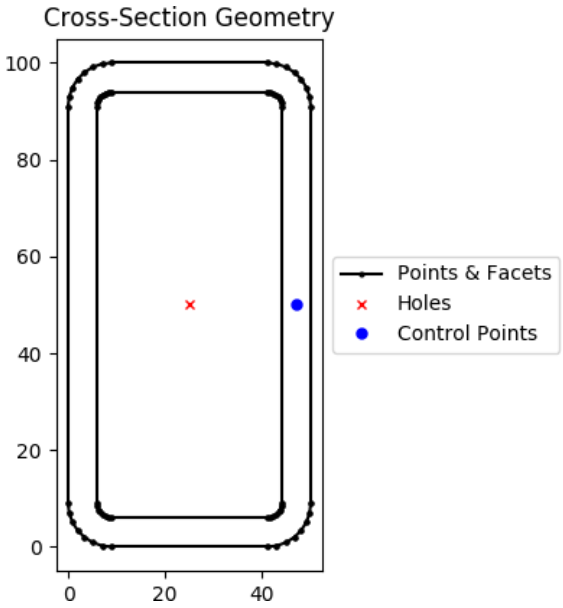


Fig. 15: RHS geometry.

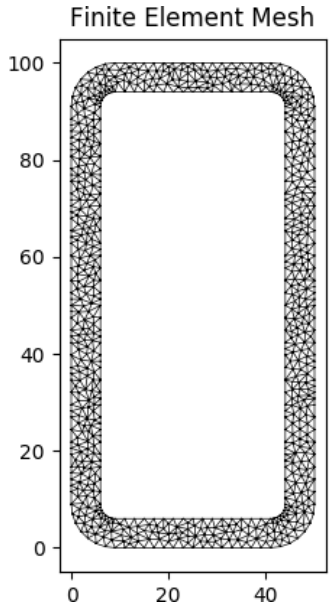


Fig. 16: Mesh generated from the above geometry.

- **d** (*float*) – Depth of the I-section
- **b** (*float*) – Width of the I-section
- **t_f** (*float*) – Flange thickness of the I-section
- **t_w** (*float*) – Web thickness of the I-section
- **r** (*float*) – Root radius of the I-section
- **n_r** (*int*) – Number of points discretising the root radius
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates an I-section with a depth of 203, a width of 133, a flange thickness of 7.8, a web thickness of 5.8 and a root radius of 8.9, using 16 points to discretise the root radius. A mesh is generated with a maximum triangular area of 3.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.ISection(d=203, b=133, t_f=7.8, t_w=5.8, r=8.9, n_
↪r=16)
mesh = geometry.create_mesh(mesh_sizes=[3.0])
```

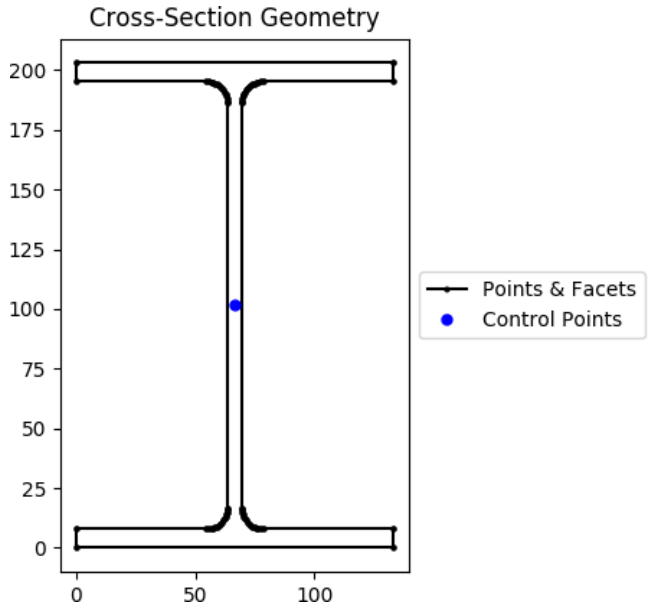


Fig. 17: I-section geometry.

MonoISection Class

class sectionproperties.pre.sections.**MonoISection**(*d, b_t, b_b, t_fb, t_ft, t_w, r, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a monosymmetric I-section centered at *(max(b_t, b_b)/2, d/2)*, with depth *d*, top flange width *b_t*, bottom flange width *b_b*, top flange thickness *t_ft*, top flange thickness *t_fb*, web thickness *t_w*, and root radius *r*, using *n_r* points to construct the root radius.

Parameters

- **d** (*float*) – Depth of the I-section

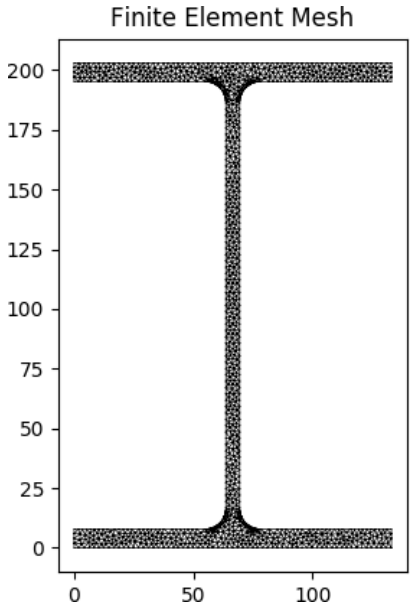


Fig. 18: Mesh generated from the above geometry.

- **b_t** (*float*) – Top flange width
- **b_b** (*float*) – Bottom flange width
- **t_ft** (*float*) – Top flange thickness of the I-section
- **t_fb** (*float*) – Bottom flange thickness of the I-section
- **t_w** (*float*) – Web thickness of the I-section
- **r** (*float*) – Root radius of the I-section
- **n_r** (*int*) – Number of points discretising the root radius
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a monosymmetric I-section with a depth of 200, a top flange width of 50, a top flange thickness of 12, a bottom flange width of 130, a bottom flange thickness of 8, a web thickness of 6 and a root radius of 8, using 16 points to discretise the root radius. A mesh is generated with a maximum triangular area of 3.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.MonoISection(
    d=200, b_t=50, b_b=130, t_ft=12, t_fb=8, t_w=6, r=8, n_r=16
)
mesh = geometry.create_mesh(mesh_sizes=[3.0])
```

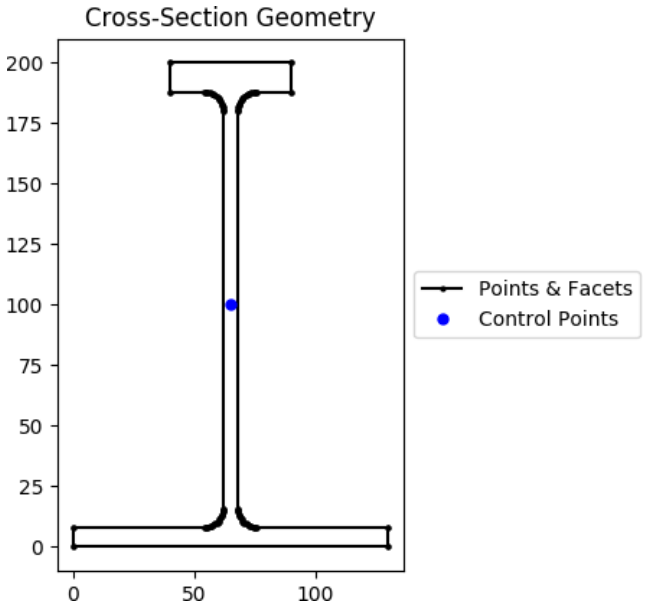


Fig. 19: I-section geometry.

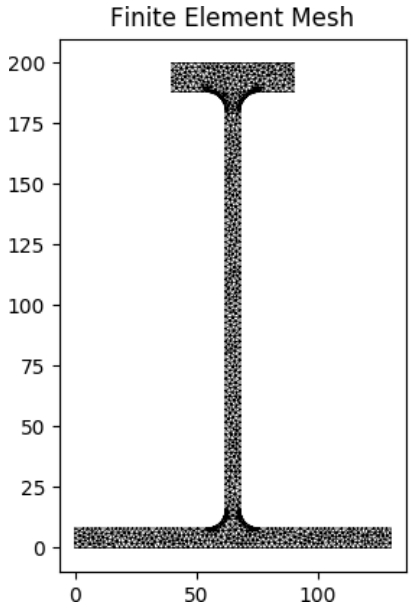


Fig. 20: Mesh generated from the above geometry.

TaperedFlangeISection Class

class sectionproperties.pre.sections.**TaperedFlangeISection**(*d, b, t_f, t_w, r_r, r_f, alpha, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a Tapered Flange I-section centered at *(b/2, d/2)*, with depth *d*, width *b*, mid-flange thickness *t_f*, web thickness *t_w*, root radius *r_r*, flange radius *r_f* and flange angle *alpha*, using *n_r* points to construct the radii.

Parameters

- **d** (*float*) – Depth of the Tapered Flange I-section
- **b** (*float*) – Width of the Tapered Flange I-section
- **t_f** (*float*) – Mid-flange thickness of the Tapered Flange I-section (measured at the point equidistant from the face of the web to the edge of the flange)
- **t_w** (*float*) – Web thickness of the Tapered Flange I-section
- **r_r** (*float*) – Root radius of the Tapered Flange I-section
- **r_f** (*float*) – Flange radius of the Tapered Flange I-section
- **alpha** (*float*) – Flange angle of the Tapered Flange I-section (degrees)
- **n_r** (*int*) – Number of points discretising the radii
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a Tapered Flange I-section with a depth of 588, a width of 191, a mid-flange thickness of 27.2, a web thickness of 15.2, a root radius of 17.8, a flange radius of 8.9 and a flange angle of 8°, using 16 points to discretise the radii. A mesh is generated with a maximum triangular area of 20.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.TaperedFlangeISection(
    d=588, b=191, t_f=27.2, t_w=15.2, r_r=17.8, r_f=8.9, alpha=8, n_r=16
)
mesh = geometry.create_mesh(mesh_sizes=[20.0])
```

PfcSection Class

class sectionproperties.pre.sections.**PfcSection**(*d, b, t_f, t_w, r, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a PFC section with the bottom left corner at the origin *(0, 0)*, with depth *d*, width *b*, flange thickness *t_f*, web thickness *t_w* and root radius *r*, using *n_r* points to construct the root radius.

Parameters

- **d** (*float*) – Depth of the PFC section
- **b** (*float*) – Width of the PFC section

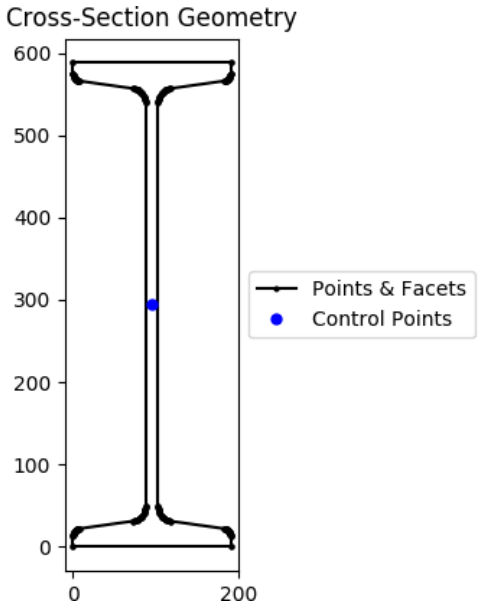


Fig. 21: I-section geometry.

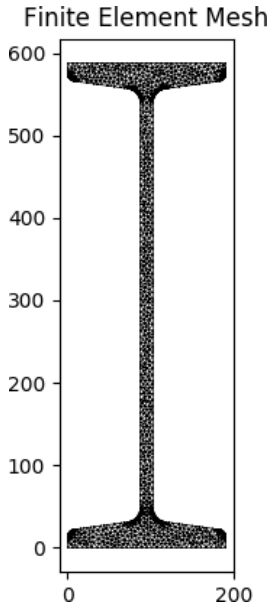


Fig. 22: Mesh generated from the above geometry.

- **t_f** (*float*) – Flange thickness of the PFC section
- **t_w** (*float*) – Web thickness of the PFC section
- **r** (*float*) – Root radius of the PFC section
- **n_r** (*int*) – Number of points discretising the root radius
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a PFC section with a depth of 250, a width of 90, a flange thickness of 15, a web thickness of 8 and a root radius of 12, using 8 points to discretise the root radius. A mesh is generated with a maximum triangular area of 5.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.PfcSection(d=250, b=90, t_f=15, t_w=8, r=12, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[5.0])
```

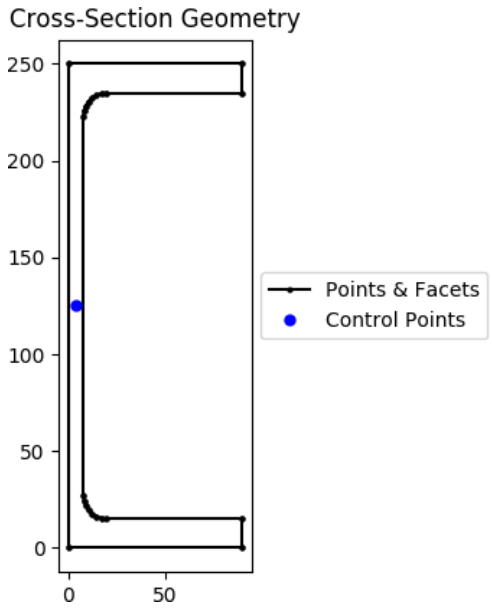


Fig. 23: PFC geometry.

TaperedFlangeChannel Class

class sectionproperties.pre.sections.**TaperedFlangeChannel**(*d, b, t_f, t_w, r_r, r_f, alpha, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a Tapered Flange Channel section with the bottom left corner at the origin *(0, 0)*, with depth *d*, width *b*, mid-flange thickness *t_f*, web thickness *t_w*, root radius *r_r*, flange radius *r_f* and flange angle *alpha*, using *n_r* points to construct the radii.

Parameters

- **d** (*float*) – Depth of the Tapered Flange Channel section

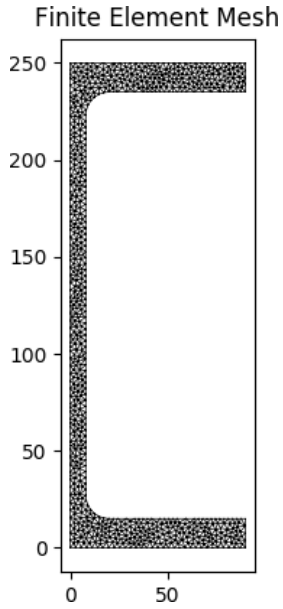


Fig. 24: Mesh generated from the above geometry.

- **b** (*float*) – Width of the Tapered Flange Channel section
- **t_f** (*float*) – Mid-flange thickness of the Tapered Flange Channel section (measured at the point equidistant from the face of the web to the edge of the flange)
- **t_w** (*float*) – Web thickness of the Tapered Flange Channel section
- **r_r** (*float*) – Root radius of the Tapered Flange Channel section
- **r_f** (*float*) – Flange radius of the Tapered Flange Channel section
- **alpha** (*float*) – Flange angle of the Tapered Flange Channel section (degrees)
- **n_r** (*int*) – Number of points discretising the radii
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a Tapered Flange Channel section with a depth of 10, a width of 3.5, a mid-flange thickness of 0.575, a web thickness of 0.475, a root radius of 0.575, a flange radius of 0.4 and a flange angle of 8°, using 16 points to discretise the radii. A mesh is generated with a maximum triangular area of 0.02:

```
import sectionproperties.pre.sections as sections

geometry = sections.TaperedFlangeChannel(
    d=10, b=3.5, t_f=0.575, t_w=0.475, r_r=0.575, r_f=0.4, alpha=8, n_
↪r=16
)
mesh = geometry.create_mesh(mesh_sizes=[0.02])
```

TeeSection Class

class sectionproperties.pre.sections.**TeeSection**(*d, b, t_f, t_w, r, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

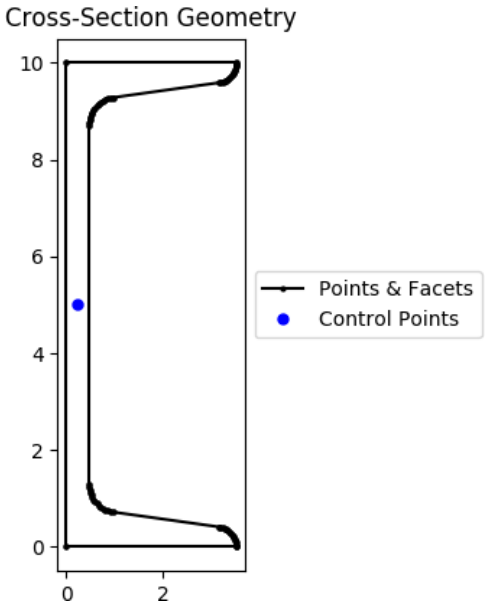


Fig. 25: I-section geometry.

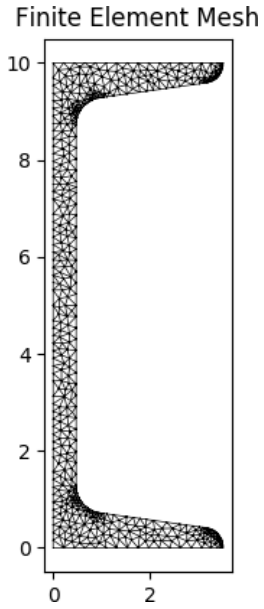


Fig. 26: Mesh generated from the above geometry.

Constructs a Tee section with the top left corner at *(0, d)*, with depth *d*, width *b*, flange thickness *t_f*, web thickness *t_w* and root radius *r*, using *n_r* points to construct the root radius.

Parameters

- **d** (*float*) – Depth of the Tee section
- **b** (*float*) – Width of the Tee section
- **t_f** (*float*) – Flange thickness of the Tee section
- **t_w** (*float*) – Web thickness of the Tee section
- **r** (*float*) – Root radius of the Tee section
- **n_r** (*int*) – Number of points discretising the root radius
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a Tee section with a depth of 200, a width of 100, a flange thickness of 12, a web thickness of 6 and a root radius of 8, using 8 points to discretise the root radius. A mesh is generated with a maximum triangular area of 3.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.TeeSection(d=200, b=100, t_f=12, t_w=6, r=8, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[3.0])
```

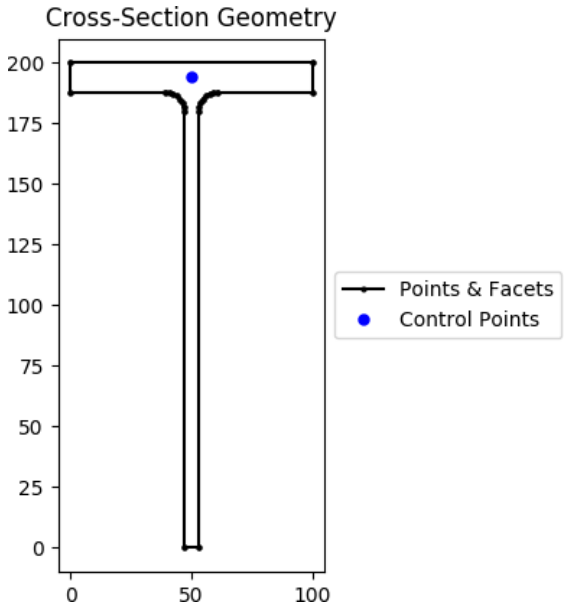


Fig. 27: Tee section geometry.

AngleSection Class

class sectionproperties.pre.sections.**AngleSection**(*d, b, t, r_r, r_t, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs an angle section with the bottom left corner at the origin *(0, 0)*, with depth *d*, width *b*, thickness *t*, root radius *r_r* and toe radius *r_t*, using *n_r* points to construct the radii.

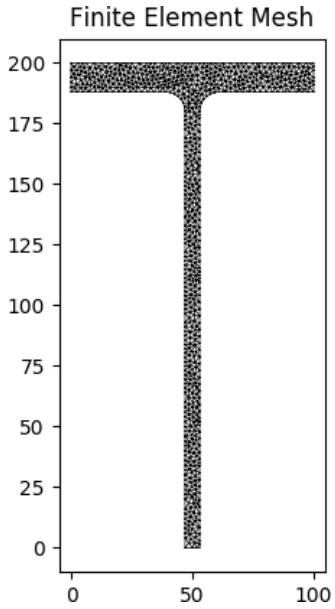


Fig. 28: Mesh generated from the above geometry.

Parameters

- **d** (*float*) – Depth of the angle section
- **b** (*float*) – Width of the angle section
- **t** (*float*) – Thickness of the angle section
- **r_r** (*float*) – Root radius of the angle section
- **r_t** (*float*) – Toe radius of the angle section
- **n_r** (*int*) – Number of points discretising the radii
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates an angle section with a depth of 150, a width of 100, a thickness of 8, a root radius of 12 and a toe radius of 5, using 16 points to discretise the radii. A mesh is generated with a maximum triangular area of 2.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.AngleSection(d=150, b=100, t=8, r_r=12, r_t=5, n_
↪r=16)
mesh = geometry.create_mesh(mesh_sizes=[2.0])
```

CeeSection Class

class sectionproperties.pre.sections.**CeeSection**(*d, b, l, t, r_out, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a Cee section with the bottom left corner at the origin *(0, 0)*, with depth *d*, width *b*, lip *l*, thickness *t* and outer radius *r_out*, using *n_r* points to construct the radius. If the outer radius is less than the thickness of the Cee Section, the inner radius is set to zero.

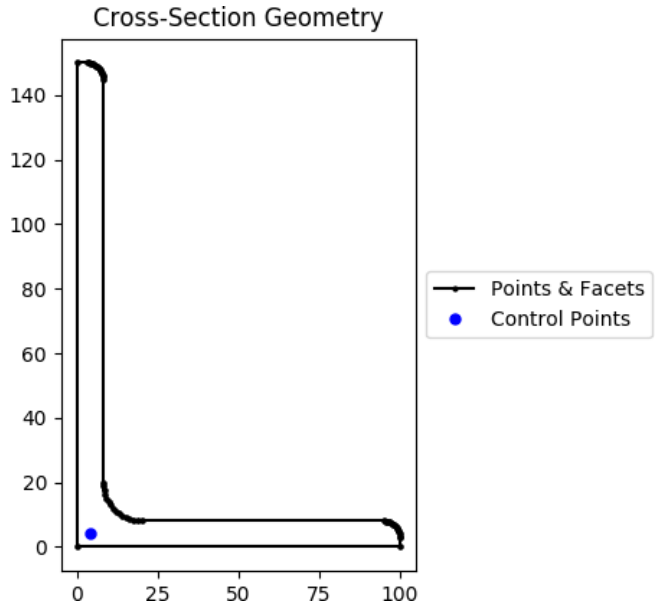
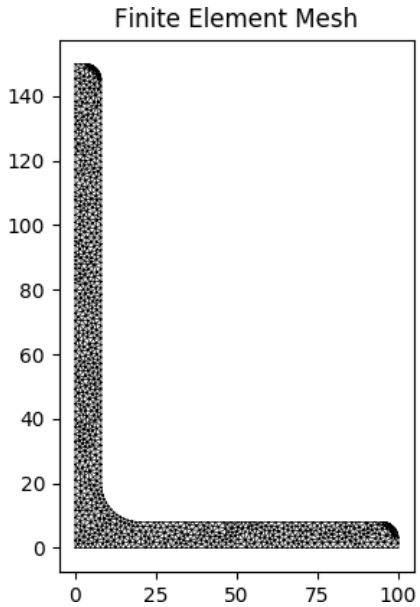


Fig. 29: Angle section geometry.



Parameters

- **d** (*float*) – Depth of the Cee section
- **b** (*float*) – Width of the Cee section
- **l** (*float*) – Lip of the Cee section
- **t** (*float*) – Thickness of the Cee section
- **r_out** (*float*) – Outer radius of the Cee section
- **n_r** (*int*) – Number of points discretising the outer radius
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

Raises **Exception** – Lip length must be greater than the outer radius

The following example creates a Cee section with a depth of 125, a width of 50, a lip of 30, a thickness of 1.5 and an outer radius of 6, using 8 points to discretise the radius. A mesh is generated with a maximum triangular area of 0.25:

```
import sectionproperties.pre.sections as sections

geometry = sections.CeeSection(d=125, b=50, l=30, t=1.5, r_out=6, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[0.25])
```

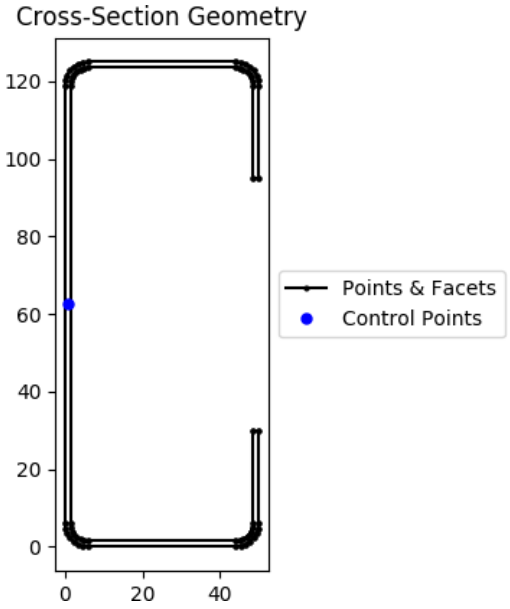


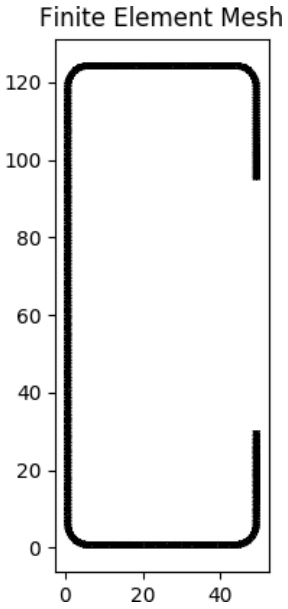
Fig. 30: Cee section geometry.

ZedSection Class

class sectionproperties.pre.sections.**ZedSection**(*d, b_l, b_r, l, t, r_out, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a Zed section with the bottom left corner at the origin *(0, 0)*, with depth *d*, left flange width *b_l*, right flange width *b_r*, lip *l*, thickness *t* and outer radius *r_out*, using *n_r* points to construct the radius. If the outer radius is less than the thickness of the Zed Section, the inner radius is set to zero.



Parameters

- **d** (*float*) – Depth of the Zed section
- **b_l** (*float*) – Left flange width of the Zed section
- **b_r** (*float*) – Right flange width of the Zed section
- **l** (*float*) – Lip of the Zed section
- **t** (*float*) – Thickness of the Zed section
- **r_out** (*float*) – Outer radius of the Zed section
- **n_r** (*int*) – Number of points discretising the outer radius
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

Raises **Exception** – Lip length must be greater than the outer radius

The following example creates a Zed section with a depth of 100, a left flange width of 40, a right flange width of 50, a lip of 20, a thickness of 1.2 and an outer radius of 5, using 8 points to discretise the radius. A mesh is generated with a maximum triangular area of 0.15:

```
import sectionproperties.pre.sections as sections

geometry = sections.ZedSection(d=100, b_l=40, b_r=50, l=20, t=1.2, r_
↪out=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[0.15])
```

CruciformSection Class

class sectionproperties.pre.sections.**CruciformSection**(*d, b, t, r, n_r, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a cruciform section centered at the origin *(0, 0)*, with depth *d*, width *b*, thickness *t* and root radius *r*, using *n_r* points to construct the root radius.

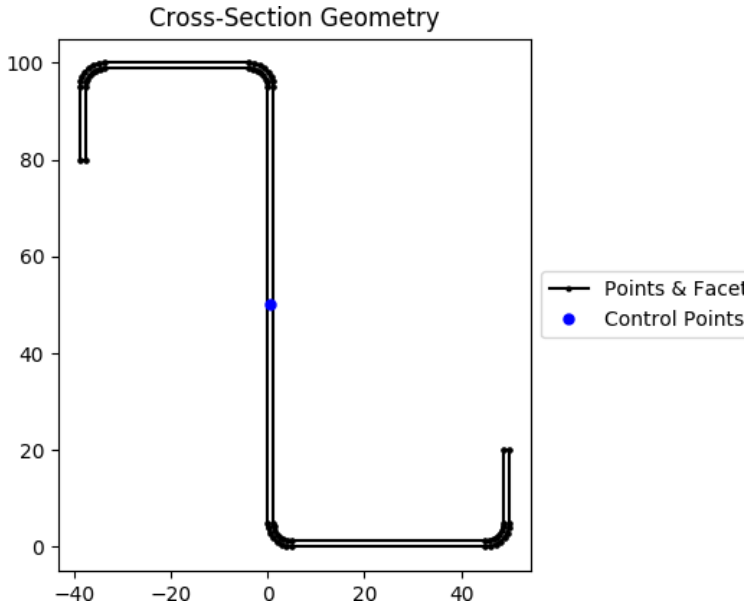
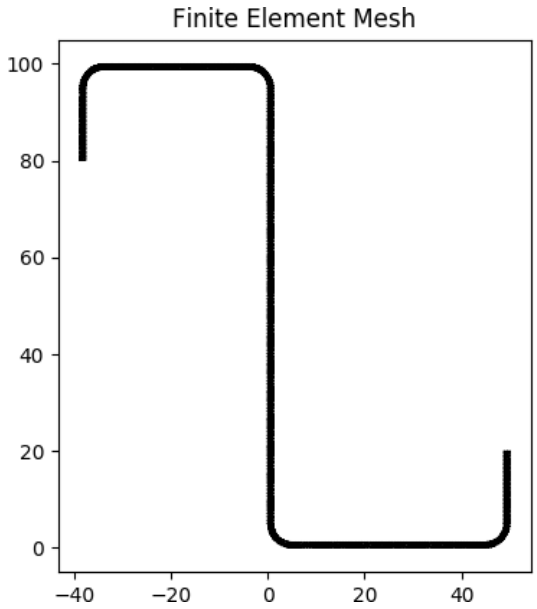


Fig. 31: Zed section geometry.



Parameters

- **d** (*float*) – Depth of the cruciform section
- **b** (*float*) – Width of the cruciform section
- **t** (*float*) – Thickness of the cruciform section
- **r** (*float*) – Root radius of the cruciform section
- **n_r** (*int*) – Number of points discretising the root radius
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a cruciform section with a depth of 250, a width of 175, a thickness of 12 and a root radius of 16, using 16 points to discretise the radius. A mesh is generated with a maximum triangular area of 5.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.CruciformSection(d=250, b=175, t=12, r=16, n_r=16)
mesh = geometry.create_mesh(mesh_sizes=[5.0])
```

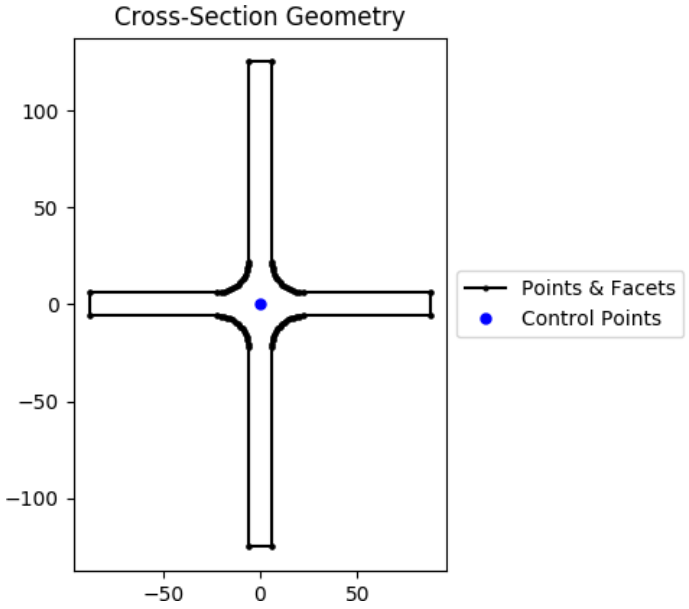


Fig. 32: Cruciform section geometry.

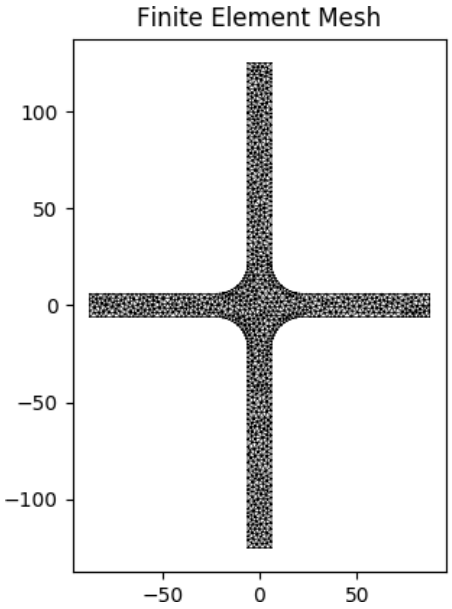
PolygonSection Class

class sectionproperties.pre.sections.**PolygonSection**(*d, t, n_sides, r_in=0, n_r=1, rot=0, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a regular hollow polygon section centered at *(0, 0)*, with a pitch circle diameter of bounding polygon *d*, thickness *t*, number of sides *n_sides* and an optional inner radius *r_in*, using *n_r* points to construct the inner and outer radii (if radii is specified).

Parameters



- **d** (*float*) – Pitch circle diameter of the outer bounding polygon (i.e. diameter of circle that passes through all vertices of the outer polygon)
- **t** (*float*) – Thickness of the polygon section wall
- **r_in** (*float*) – Inner radius of the polygon corners. By default, if not specified, a polygon with no corner radii is generated.
- **n_r** (*int*) – Number of points discretising the inner and outer radii, ignored if no inner radii is specified
- **rot** – Initial counterclockwise rotation in degrees. By default bottom face is aligned with x axis.
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

Raises **Exception** – Number of sides in polygon must be greater than or equal to 3

The following example creates an Octagonal section (8 sides) with a diameter of 200, a thickness of 6 and an inner radius of 20, using 12 points to discretise the inner and outer radii. A mesh is generated with a maximum triangular area of 5:

```
import sectionproperties.pre.sections as sections

geometry = sections.PolygonSection(d=200, t=6, n_sides=8, r_in=20, n_
↪r=12)
mesh = geometry.create_mesh(mesh_sizes=[5])
```

BoxGirderSection Class

class sectionproperties.pre.sections.**BoxGirderSection**(*d, b_t, b_b, t_ft, t_fb, t_w, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a Box Girder section centered at at *(max(b_t, b_b)/2, d/2)*, with depth *d*, top width *b_t*, bottom width *b_b*, top flange thickness *t_ft*, bottom flange thickness *t_fb* and web thickness *t_w*.

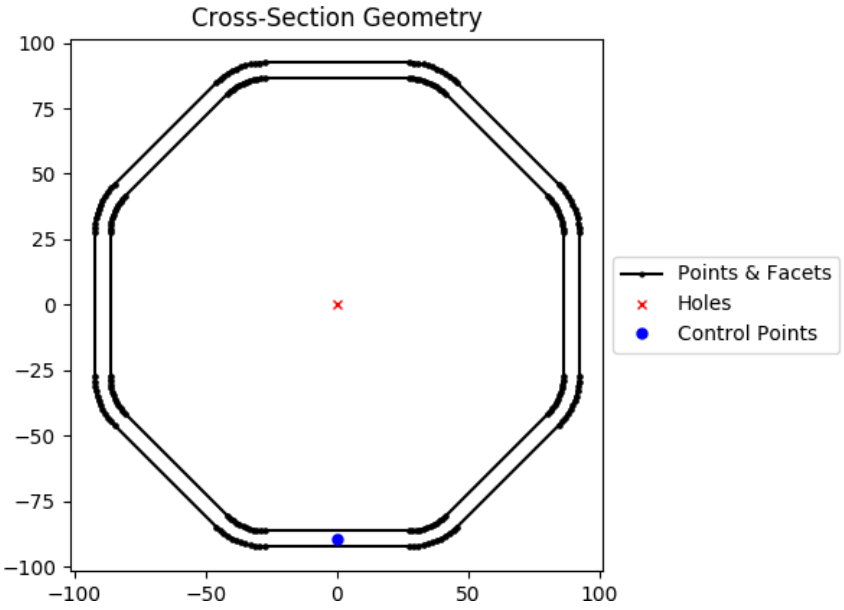


Fig. 33: Octagonal section geometry.

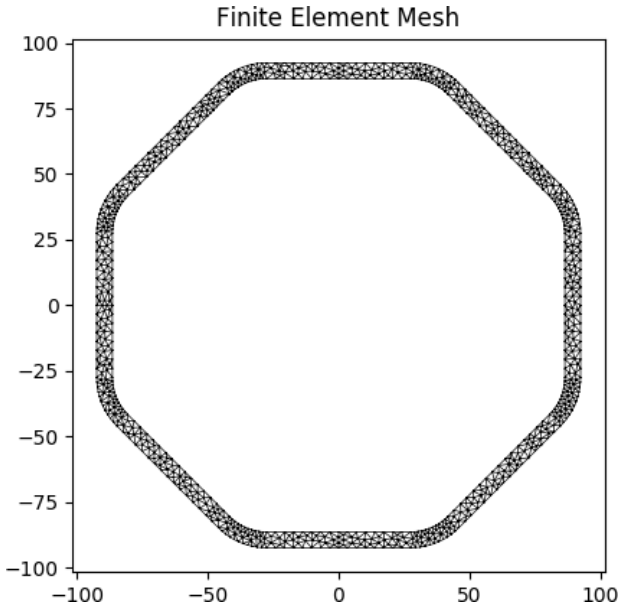


Fig. 34: Mesh generated from the above geometry.

Parameters

- **d** (*float*) – Depth of the Box Girder section
- **b_t** (*float*) – Top width of the Box Girder section
- **b_b** (*float*) – Bottom width of the Box Girder section
- **t_ft** (*float*) – Top lange thickness of the Box Girder section
- **t_fb** (*float*) – Bottom flange thickness of the Box Girder section
- **t_w** (*float*) – Web thickness of the Box Girder section
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a Box Gider section with a depth of 1200, a top width of 1200, a bottom width of 400, a top flange thickness of 16, a bottom flange thickness of 12 and a web thickness of 8. A mesh is generated with a maximum triangular area of 5.0:

```
import sectionproperties.pre.sections as sections

geometry = sections.BoxGirderSection(d=1200, b_t=1200, b_b=400, t_ft=100,
↪ t_fb=80, t_w=50)
mesh = geometry.create_mesh(mesh_sizes=[200.0])
```

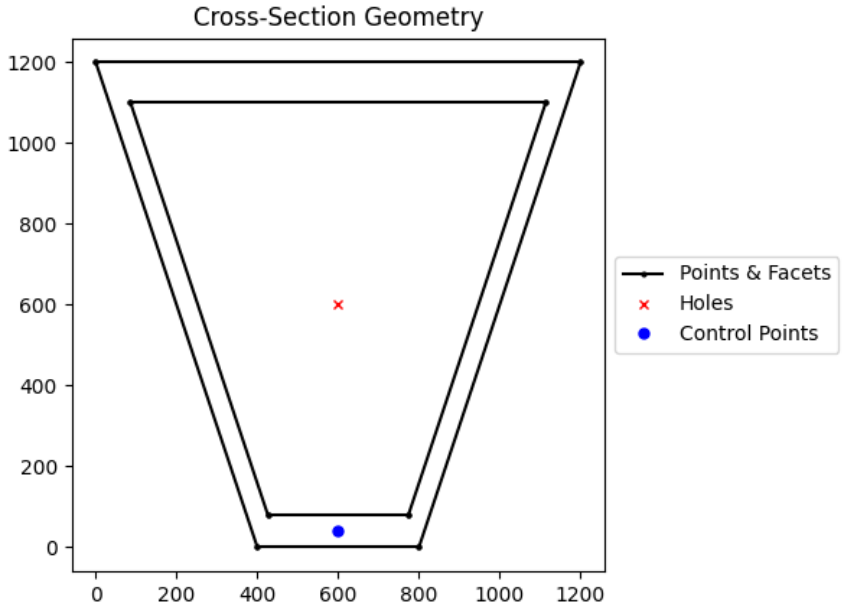


Fig. 35: Box Girder geometry.

MergedSection Class

class sectionproperties.pre.sections.**MergedSection**(*sections*)

Bases: *sectionproperties.pre.sections.Geometry*

Merges a number of section geometries into one geometry. Note that for the meshing algorithm to work, there needs to be connectivity between all regions of the provided geometries. Overlapping of geometries is permitted.

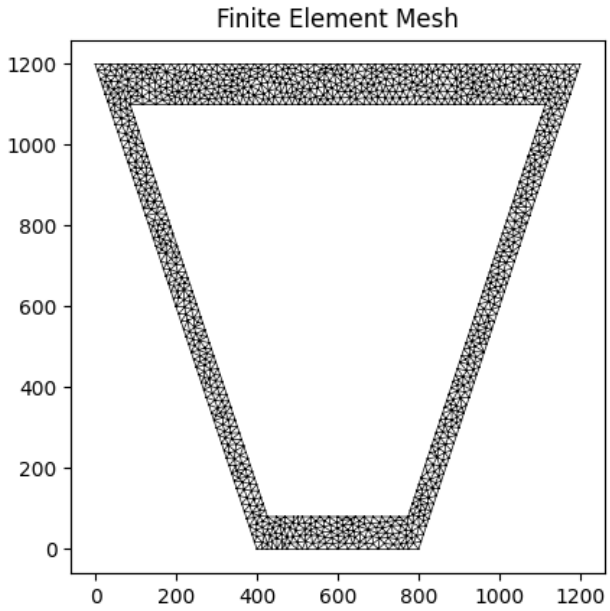


Fig. 36: Mesh generated from the above geometry.

Parameters **sections** (list[*Geometry*]) – A list of geometry objects to merge into one *Geometry* object

The following example creates a combined cross-section with a 150x100x6 RHS placed on its side on top of a 200UB25.4. A mesh is generated with a maximum triangle size of 5.0 for the I-section and 2.5 for the RHS:

```
import sectionproperties.pre.sections as sections

isection = sections.ISection(d=203, b=133, t_f=7.8, t_w=5.8, r=8.9, n_
↪r=8)
box = sections.Rhs(d=100, b=150, t=6, r_out=15, n_r=8, shift=[-8.5, 203])

geometry = sections.MergedSection([isection, box])
geometry.clean_geometry()
mesh = geometry.create_mesh(mesh_sizes=[5.0, 2.5])
```

7.1.2 *pre* Module

Material Class

class sectionproperties.pre.pre.**Material**(*name*, *elastic_modulus*, *poissons_ratio*, *yield_strength*, *color='w'*)

Bases: object

Class for structural materials.

Provides a way of storing material properties related to a specific material. The color can be a multitude of different formats, refer to https://matplotlib.org/api/colors_api.html and https://matplotlib.org/examples/color/named_colors.html for more information.

Parameters

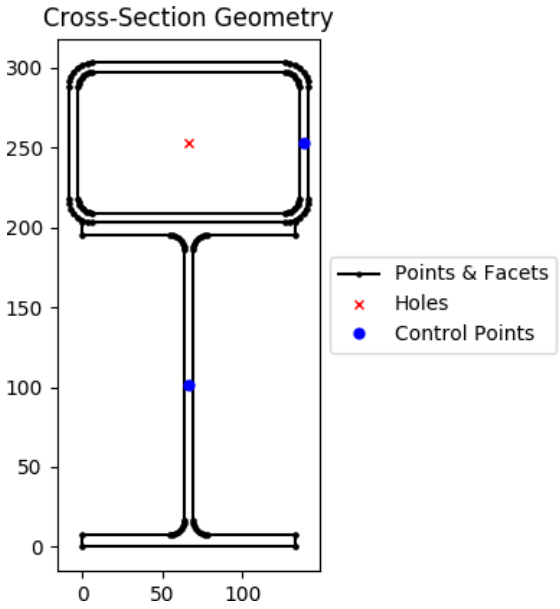
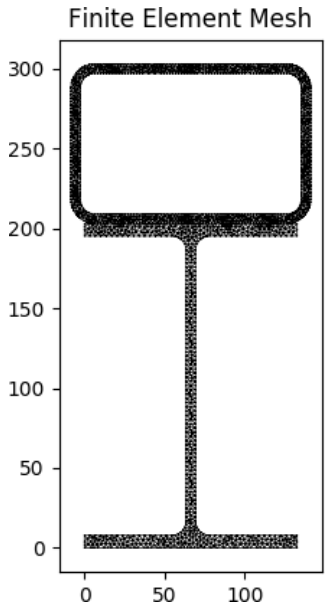


Fig. 37: Merged section geometry.



- **name** (*string*) – Material name
- **elastic_modulus** (*float*) – Material modulus of elasticity
- **poissons_ratio** (*float*) – Material Poisson's ratio
- **yield_strength** (*float*) – Material yield strength
- **color** (matplotlib.colors) – Material color for rendering

Variables

- **name** (*string*) – Material name
- **elastic_modulus** (*float*) – Material modulus of elasticity
- **poissons_ratio** (*float*) – Material Poisson's ratio
- **shear_modulus** (*float*) – Material shear modulus, derived from the elastic modulus and Poisson's ratio assuming an isotropic material
- **yield_strength** (*float*) – Material yield strength
- **color** (matplotlib.colors) – Material color for rendering

The following example creates materials for concrete, steel and timber:

```
from sectionproperties.pre.pre import Material

concrete = Material(
    name='Concrete', elastic_modulus=30.1e3, poissons_ratio=0.2, yield_
↪strength=32,
        color='lightgrey'
)
steel = Material(
    name='Steel', elastic_modulus=200e3, poissons_ratio=0.3, yield_
↪strength=500,
        color='grey'
)
timber = Material(
    name='Timber', elastic_modulus=8e3, poissons_ratio=0.35, yield_
↪strength=20,
        color='burlywood'
)
```

GeometryCleaner Class

class sectionproperties.pre.pre.**GeometryCleaner**(*geometry*, *verbose*)

Bases: object

Class for cleaning *Geometry* objects.

Parameters

- **geometry** (*Geometry*) – Geometry object to clean
- **verbose** (*bool*) – If set to true, information related to the geometry cleaning process is printed to the terminal.

Provides methods to clean various aspects of the geometry including:

- Zipping nodes - Find nodes that are close together (relative and absolute tolerance) and deletes one of the nodes and rejoins the facets to the remaining node.

- Removing zero length facets - Removes facets that start and end at the same point.
- Remove duplicate facets - Removes facets that have the same starting and ending point as an existing facet.
- Removing overlapping facets - Searches for facets that overlap each other, given a tolerance angle, and reconstructs a unique set of facets along the overlapping region.
- Remove unused points - Removes points that are not connected to any facets.
- Intersect facets - Searches for intersections between two facets and adds the intersection point to the points list and splits the intersected facets.

Note that a geometry cleaning method is provided to all *Geometry* objects.

Variables

- **geometry** (*Geometry*) – Geometry object to clean
- **verbose** (*bool*) – If set to true, information related to the geometry cleaning process is printed to the terminal.

The following example creates a back-to-back 200PFC geometry, rotates the geometry by 30 degrees, and cleans the geometry before meshing:

```
import sectionproperties.pre.sections as sections

pfc_right = sections.PfcSection(d=203, b=133, t_f=7.8, t_w=5.8, r=8.9, n_
↪r=8)
pfc_left = sections.PfcSection(d=203, b=133, t_f=7.8, t_w=5.8, r=8.9, n_
↪r=8)
pfc_left.mirror_section(axis='y', mirror_point=[0, 0])
geometry = sections.MergedSection([pfc_left, pfc_right])
geometry.rotate_section(angle=30)
geometry.clean_geometry(verbose=True)
mesh = geometry.create_mesh(mesh_sizes=[5, 5])
```

Warning: If the geometry were not cleaned in the previous example, the meshing algorithm would crash (most likely return a segment error). Cleaning the geometry is always recommended when creating a merged section, which may result in overlapping or intersecting facets, or duplicate nodes.

clean_geometry()
Performs a full geometry clean on the *geometry* object.

intersect_facets()
Searches through all facet combinations and finds facets that intersect each other. The intersection point is added and the facets rebuilt.

is_duplicate_facet(*fct1*, *fct2*)
Checks to see if to facets are duplicates.

Parameters
- **fct1** (*list[int, int]*) – First facet to compare
- **fct2** (*list[int, int]*) – Second facet to compare

Returns Whether or not the facets are identical

Return type bool

is_intersect(*p*, *q*, *r*, *s*)
Determines if the line segment p->p+r intersects q->q+s. Implements Gareth Rees's answer:

https://stackoverflow.com/questions/563198.

Parameters

- **p** (numpy.ndarray [float, float]) – Starting point of the first line segment
- **q** (numpy.ndarray [float, float]) – Starting point of the second line segment
- **r** (numpy.ndarray [float, float]) – Vector of the first line segment
- **s** (numpy.ndarray [float, float]) – Vector of the second line segment

Returns The intersection point of the line segments. If there is no intersection, returns None.

Return type numpy.ndarray [float, float]

is_overlap(*p*, *q*, *r*, *s*, *fct1*, *fct2*)

Determines if the line segment p->p+r overlaps q->q+s. Implements Gareth Rees's answer: https://stackoverflow.com/questions/563198.

Parameters

- **p** (numpy.ndarray [float, float]) – Starting point of the first line segment
- **q** (numpy.ndarray [float, float]) – Starting point of the second line segment
- **r** (numpy.ndarray [float, float]) – Vector of the first line segment
- **s** (numpy.ndarray [float, float]) – Vector of the second line segment
- **fct1** – sadkjas;dkas;dj

Returns A list containing the points required for facet rebuilding. If there is no rebuild to be done, returns None.

Return type list[list[float, float]]

remove_duplicate_facets()

Searches through all facets and removes facets that are duplicates, independent of the point order.

remove_overlapping_facets()

Searches through all facet combinations and fixes facets that overlap within a tolerance.

remove_point_id(*point_id*)

Removes point point_id from the points list and renumbers the references to points after point_id in the facet list.

Parameters **point_id** (*int*) – Index of point to be removed

remove_unused_points()

Searches through all facets and removes points that are not connected to any facets.

remove_zero_length_facets()

Searches through all facets and removes those that have the same starting and ending point.

replace_point_id(*id_old*, *id_new*)

Searches all facets and replaces references to point id_old with id_new.

Parameters

- **id_old** (*int*) – Point index to be replaced
- **id_new** (*int*) – Point index to replace point id_old

zip_points(*atol=1e-08*, *rtol=1e-05*)

Zips points that are close to each other. Searches through the point list and merges two points if there are deemed to be sufficiently close. The average value of the coordinates is used for the new point. One of the points is deleted from the point list and the facet list is updated to remove references to the old points and renumber the remaining point indices in the facet list.

Parameters

- **atol** (*float*) – Absolute tolerance for point zipping
- **rtol** (*float*) – Relative tolerance (to geometry extents) for point zipping

pre Functions

sectionproperties.pre.pre.**create_mesh**(*points*, *facets*, *holes*, *control_points*, *mesh_sizes*)

Creates a quadratic triangular mesh using the meshpy module, which utilises the code 'Triangle', by Jonathan Shewchuk.

Parameters

- **points** (*list[list[int, int]]*) – List of points *(x, y)* defining the vertices of the cross-section
- **facets** – List of point index pairs *(p1, p2)* defining the edges of the cross-section
- **holes** (*list[list[float, float]]*) – List of points *(x, y)* defining the locations of holes within the cross-section. If there are no holes, provide an empty list [].
- **control_points** (*list[list[float, float]]*) – A list of points *(x, y)* that define different regions of the cross-section. A control point is an arbitrary point within a region enclosed by facets.
- **mesh_sizes** (*list[float]*) – List of maximum element areas for each region defined by a control point

Returns Object containing generated mesh data

Return type meshpy.triangle.MeshInfo

7.1.3 *offset* Module

sectionproperties.pre.offset.**offset_perimeter**(*geometry*, *offset*, *side='left'*, *plot_offset=False*)

Offsets the perimeter of a geometry of a *Geometry* object by a certain distance. Note that the perimeter facet list must be entered in a consecutive order.

Parameters

- **geometry** (*Geometry*) – Cross-section geometry object
- **offset** (*float*) – Offset distance for the perimeter
- **side** (*string*) – Side of the perimeter offset, either 'left' or 'right'. E.g. 'left' for a counter-clockwise offsets the perimeter inwards.
- **plot_offset** (*bool*) – If set to True, generates a plot comparing the old and new geometry

The following example 'corrodes' a 200UB25 I-section by 1.5 mm and compares a few of the section properties:

```
import sectionproperties.pre.sections as sections
from sectionproperties.pre.offset import offset_perimeter
from sectionproperties.analysis.cross_section import CrossSection

# calculate original section properties
original_geometry = sections.ISection(d=203, b=133, t_f=7.8, t_w=5.8, r=8.9, n_
↪r=16)
original_mesh = original_geometry.create_mesh(mesh_sizes=[3.0])
original_section = CrossSection(original_geometry, original_mesh)
original_section.calculate_geometric_properties()
original_area = original_section.get_area()
(original_ixx, _, _) = original_section.get_ic()
```

(continues on next page)

(continued from previous page)

```
# calculate corroded section properties
corroded_geometry = offset_perimeter(original_geometry, 1.5, plot_offset=True)
corroded_mesh = corroded_geometry.create_mesh(mesh_sizes=[3.0])
corroded_section = CrossSection(corroded_geometry, corroded_mesh)
corroded_section.calculate_geometric_properties()
corroded_area = corroded_section.get_area()
(corroded_ixx, _, _) = corroded_section.get_ic()

# compare section properties
print("Area reduction = {0:.2f}%".format(
    100 * (original_area - corroded_area) / original_area))
print("Ixx reduction = {0:.2f}%".format(
    100 *(original_ixx - corroded_ixx) / original_ixx))
```

The following plot is generated by the above example:

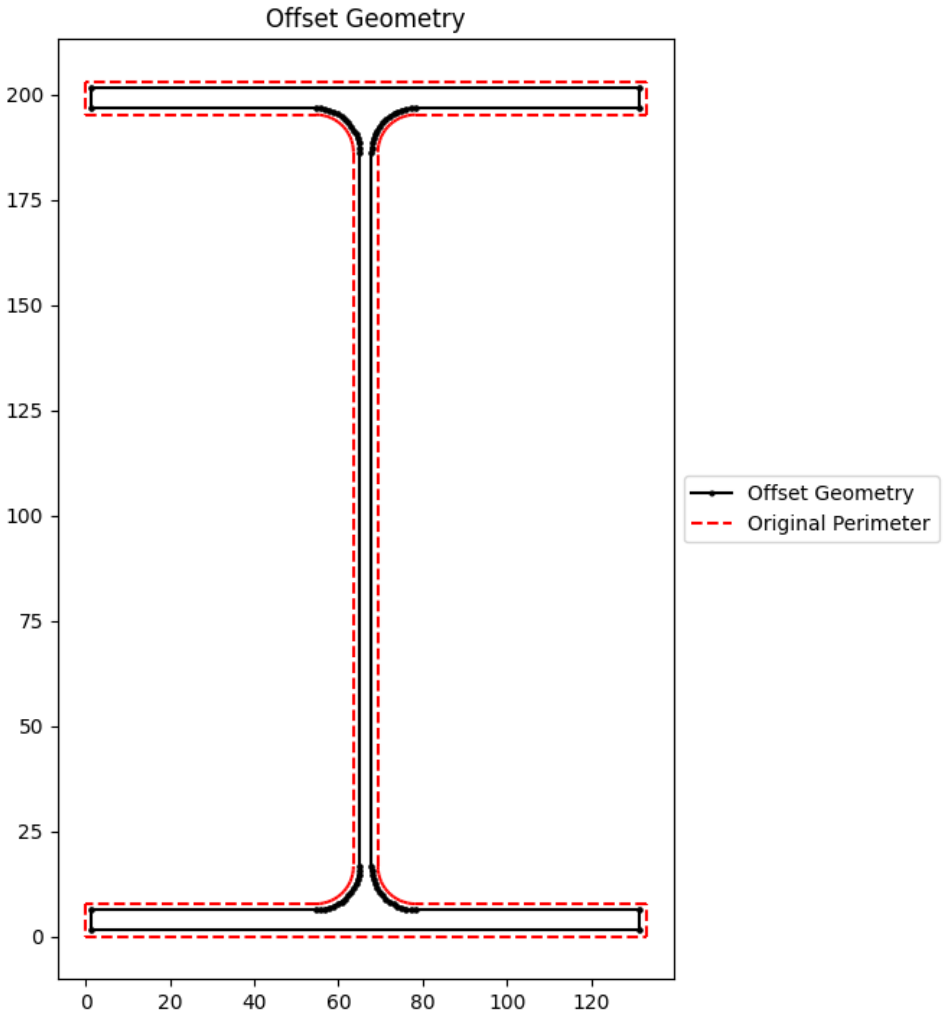


Fig. 38: 200UB25 with 1.5 mm corrosion.

The following is printed to the terminal:

```
Area reduction = 41.97%
Ixx reduction = 39.20%
```

7.1.4 *nastran_sections* Module

This module contains cross-sections as defined by Nastran and Nastran-based programs, such as MYSTRAN and ASTROS.

BARSection Class

class sectionproperties.pre.nastran_sections.**BARSection**(*DIM1, DIM2, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a BAR section with the center at the origin *(0, 0)*, with two parameters defining dimensions. See Nastran documentation[1][2][3][4][5] for definition of parameters. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Width (x) of bar
- **DIM2** (*float*) – Depth (y) of bar
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a BAR cross-section with a depth of 1.5 and width of 2.0, and generates a mesh with a maximum triangular area of 0.001:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.BARSection(DIM1=2.0, DIM2=1.5)
mesh = geometry.create_mesh(mesh_sizes=[0.001])
```

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

BOXSection Class

class sectionproperties.pre.nastran_sections.**BOXSection**(*DIM1, DIM2, DIM3, DIM4, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

[1] MSC Nastran Quick Reference Guide 2012, PBEAML - Simple Beam Cross-Section Property, pp. 2890-2894 https://simcompanion.mscsoftware.com/infocenter/index?page=content&id=DOC10351

[2] Siemens NX Nastran 12 Quick Reference Guide, PBEAML, pp. 16-59 - 16-62 https://docs.plm.automation.siemens.com/data_services/resources/nxnastran/12/help/tdoc/en_US/pdf/QRG.pdf

[3] AutoDesk Nastran Online Documentation, Nastran Reference Guide, Section 4 - Bulk Data, PBEAML http://help.autodesk.com/view/NSTRN/2018/ENU/?guid=GUID-B7044BA7-3C26-49DA-9EE7-DA7505FD4B2C

[4] Users Reference Manual for the MYSTRAN General Purpose Finite Element Structural Analysis Computer Program, Jan. 2019, Section 6.4.1.53 - PBARL, pp. 154-156 https://www.mystran.com/Executable/MYSTRAN-Users-Manual.pdf

[5] Astros Enhancements - Volume III - Astros Theoretical Manual, Section 5.1.3.2, pp. 56 https://apps.dtic.mil/dtic/tr/fulltext/u2/a308134.pdf

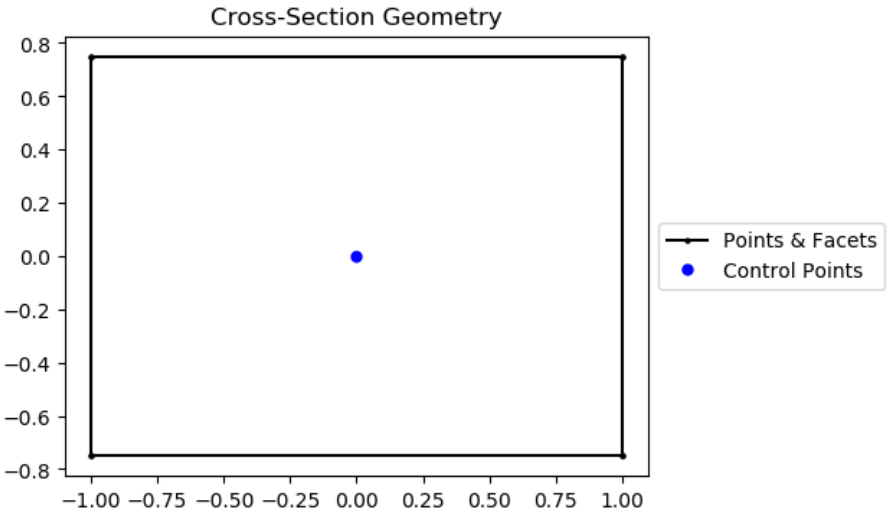


Fig. 39: BAR section geometry.

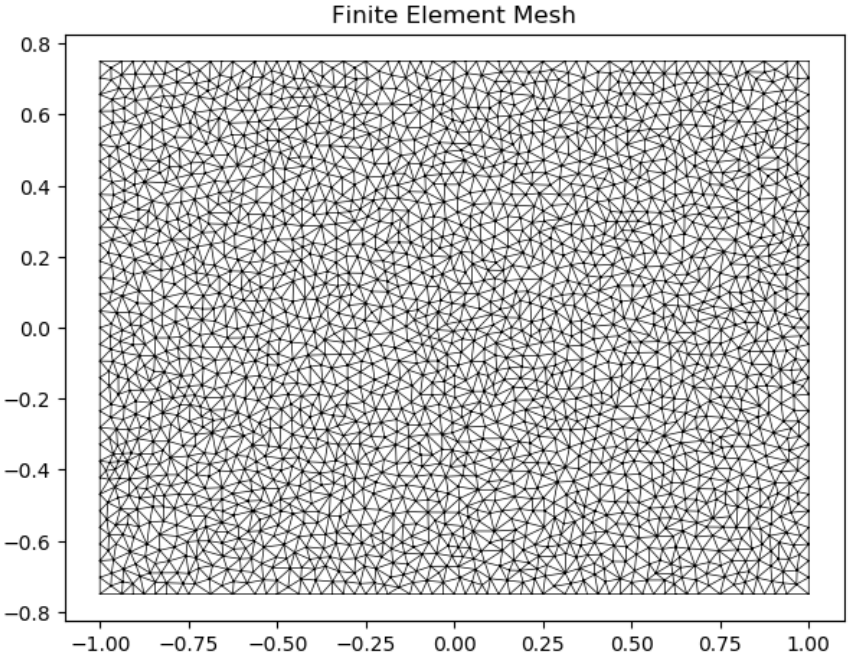


Fig. 40: Mesh generated from the above geometry.

Constructs a BOX section with the center at the origin *(0, 0)*, with four parameters defining dimensions. See Nastran documentation[12345] for definition of parameters. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Width (x) of box
- **DIM2** (*float*) – Depth (y) of box
- **DIM3** (*float*) – Thickness of box in y direction
- **DIM4** (*float*) – Thickness of box in x direction
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a BOX cross-section with a depth of 3.0 and width of 4.0, and generates a mesh with a maximum triangular area of 0.001:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.BOXSection(DIM1=4.0, DIM2=3.0, DIM3=0.375, DIM4=0.5)
mesh = geometry.create_mesh(mesh_sizes=[0.001])
```

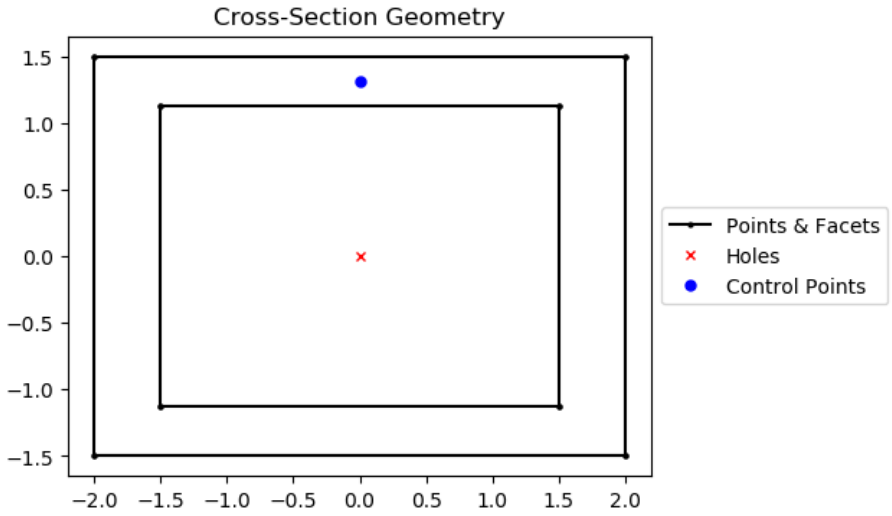


Fig. 41: BOX section geometry.

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

BOX1Section Class

class sectionproperties.pre.nastran_sections.**BOX1Section**(*DIM1, DIM2, DIM3, DIM4, DIM5, DIM6, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

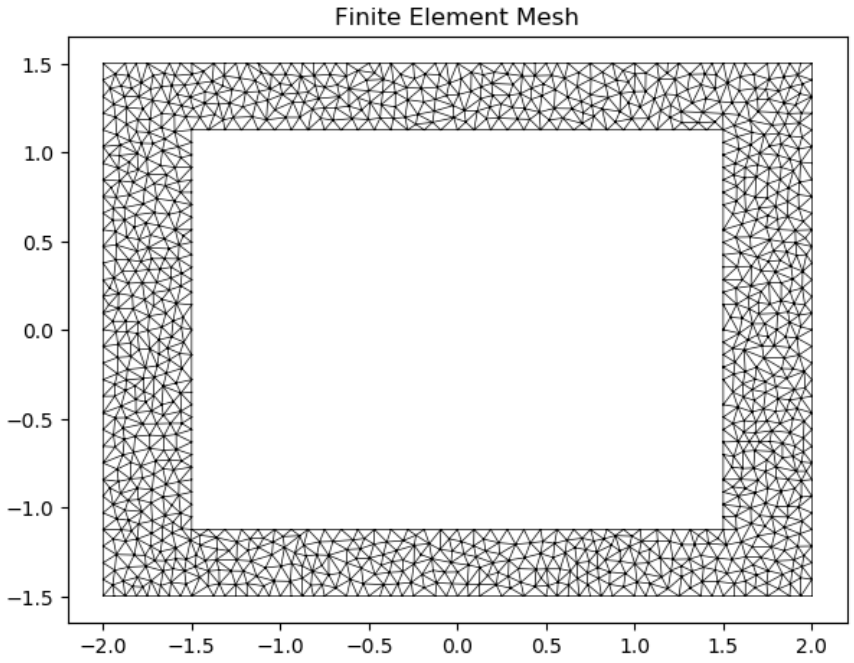


Fig. 42: Mesh generated from the above geometry.

Constructs a BOX1 section with the center at the origin *(0, 0)*, with six parameters defining dimensions. See Nastran documentation[1234] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Width (x) of box
- **DIM2** (*float*) – Depth (y) of box
- **DIM3** (*float*) – Thickness of top wall
- **DIM4** (*float*) – Thickness of bottom wall
- **DIM5** (*float*) – Thickness of left wall
- **DIM6** (*float*) – Thickness of right wall
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a BOX1 cross-section with a depth of 3.0 and width of 4.0, and generates a mesh with a maximum triangular area of 0.007:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.BOX1Section(
    DIM1=4.0, DIM2=3.0, DIM3=0.375, DIM4=0.5, DIM5=0.25, DIM6=0.75
)
mesh = geometry.create_mesh(mesh_sizes=[0.007])
```

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

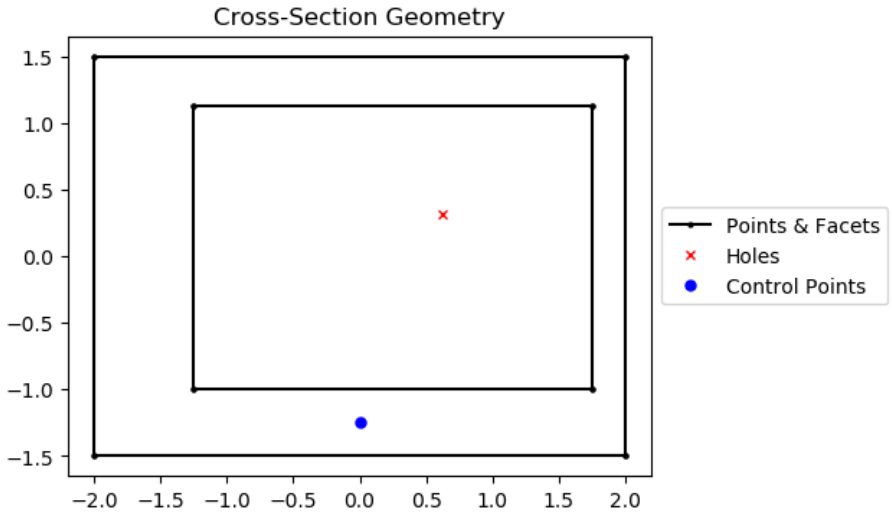


Fig. 43: BOX1 section geometry.

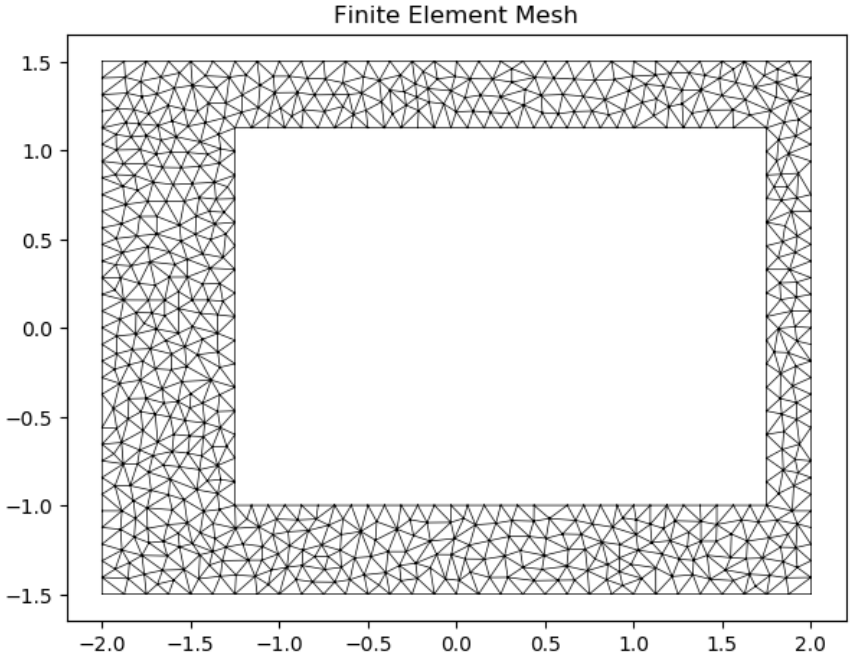


Fig. 44: Mesh generated from the above geometry.

Returns Stress evaluation points relative to shifted origin - C, D, E, F

CHANSection Class

class sectionproperties.pre.nastran_sections.**CHANSection**(*DIM1, DIM2, DIM3, DIM4, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a CHAN (C-Channel) section with the web's middle center at the origin *(0, 0)*, with four parameters defining dimensions. See Nastran documentation[1234] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Width (x) of the CHAN-section
- **DIM2** (*float*) – Depth (y) of the CHAN-section
- **DIM3** (*float*) – Thickness of web (vertical portion)
- **DIM4** (*float*) – Thickness of flanges (top/bottom portion)
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a CHAN cross-section with a depth of 4.0 and width of 2.0, and generates a mesh with a maximum triangular area of 0.008:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.CHANSection(DIM1=2.0, DIM2=4.0, DIM3=0.25, DIM4=0.5)
mesh = geometry.create_mesh(mesh_sizes=[0.008])
```

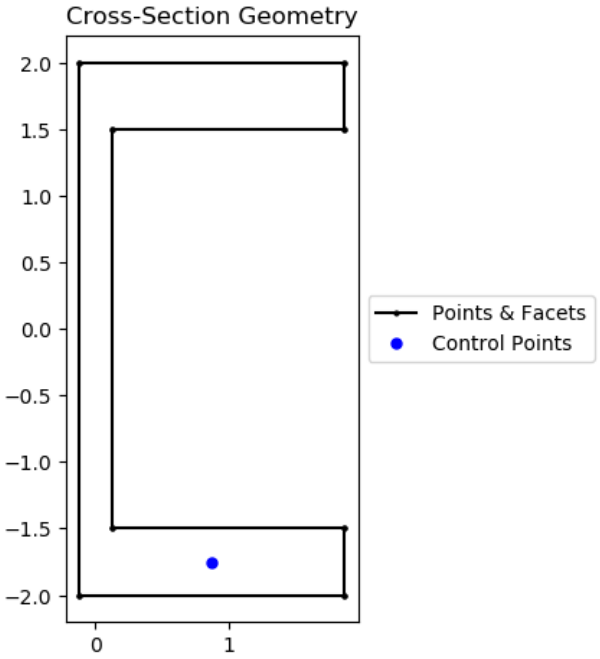


Fig. 45: CHAN section geometry.

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

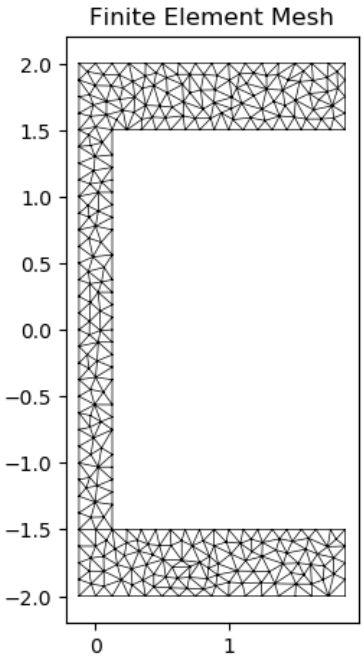


Fig. 46: Mesh generated from the above geometry.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

CHAN1Section Class

class sectionproperties.pre.nastran_sections.**CHAN1Section**(*DIM1, DIM2, DIM3, DIM4, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a CHAN1 (C-Channel) section with the web's middle center at the origin *(0, 0)*, with four parameters defining dimensions. See Nastran documentation[1234] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Width (x) of channels
- **DIM2** (*float*) – Thicknesss (x) of web
- **DIM3** (*float*) – Spacing between channels (length of web)
- **DIM4** (*float*) – Depth (y) of CHAN1-section
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a CHAN1 cross-section with a depth of 4.0 and width of 1.75, and generates a mesh with a maximum triangular area of 0.01:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.CHAN1Section(DIM1=0.75, DIM2=1.0, DIM3=3.5, DIM4=4.0)
mesh = geometry.create_mesh(mesh_sizes=[0.01])
```

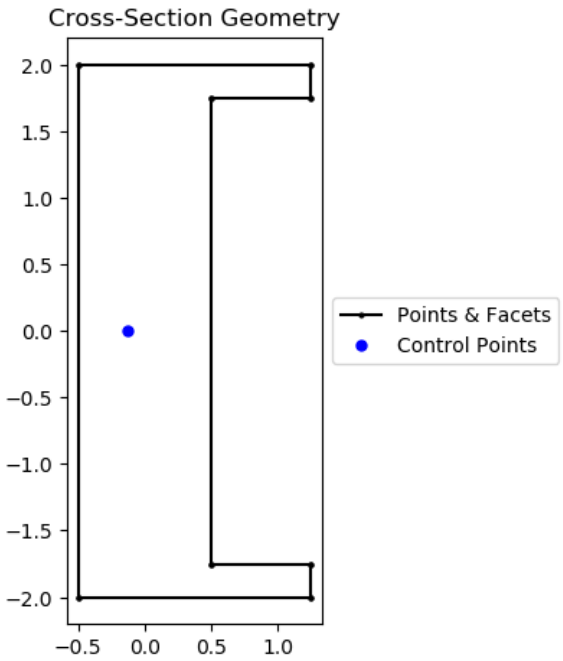


Fig. 47: CHAN1 section geometry.

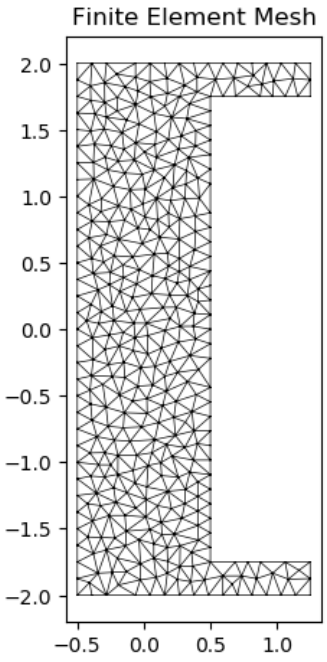


Fig. 48: Mesh generated from the above geometry.

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

CHAN2Section Class

class sectionproperties.pre.nastran_sections.**CHAN2Section**(*DIM1, DIM2, DIM3, DIM4, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a CHAN2 (C-Channel) section with the bottom web's middle center at the origin *(0, 0)*, with four parameters defining dimensions. See Nastran documentation[1234] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Thickness of channels
- **DIM2** (*float*) – Thickness of web
- **DIM3** (*float*) – Depth (y) of CHAN2-section
- **DIM4** (*float*) – Width (x) of CHAN2-section
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a CHAN2 cross-section with a depth of 2.0 and width of 4.0, and generates a mesh with a maximum triangular area of 0.01:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.CHAN2Section(DIM1=0.375, DIM2=0.5, DIM3=2.0, DIM4=4.0)
mesh = geometry.create_mesh(mesh_sizes=[0.01])
```

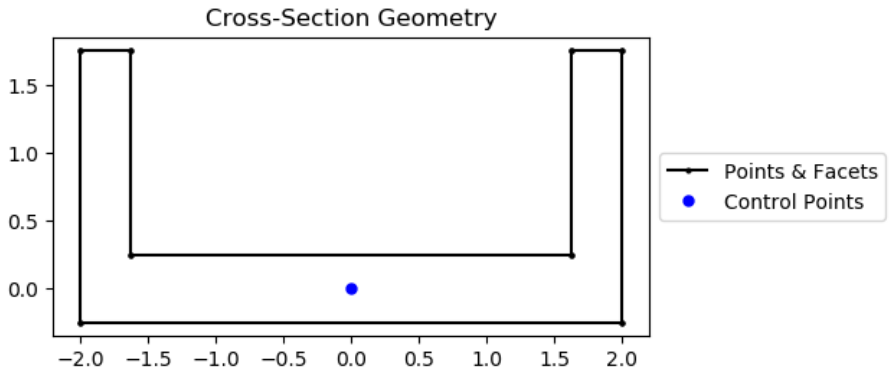


Fig. 49: CHAN2 section geometry.

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

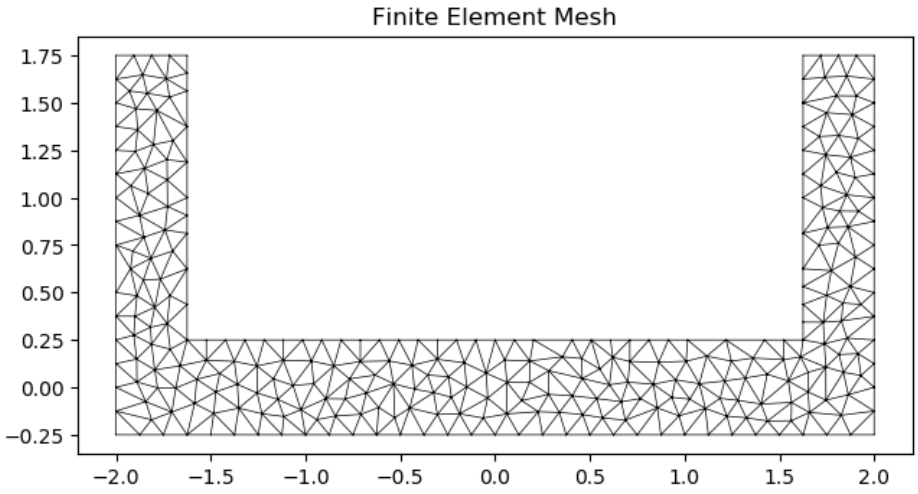


Fig. 50: Mesh generated from the above geometry.

Returns Stress evaluation points relative to shifted origin - C, D, E, F

CROSSSection Class

class sectionproperties.pre.nastran_sections.**CROSSSection**(*DIM1, DIM2, DIM3, DIM4, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs Nastran's cruciform/cross section with the intersection's middle center at the origin *(0, 0)*, with four parameters defining dimensions. See Nastran documentation[1234] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Twice the width of horizontal member protruding from the vertical center member
- **DIM2** (*float*) – Thickness of the vertical member
- **DIM3** (*float*) – Depth (y) of the CROSS-section
- **DIM4** (*float*) – Thickness of the horizontal members
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a rectangular cross-section with a depth of 3.0 and width of 1.875, and generates a mesh with a maximum triangular area of 0.008:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.CROSSSection(DIM1=1.5, DIM2=0.375, DIM3=3.0, DIM4=0.25)
mesh = geometry.create_mesh(mesh_sizes=[0.008])
```

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

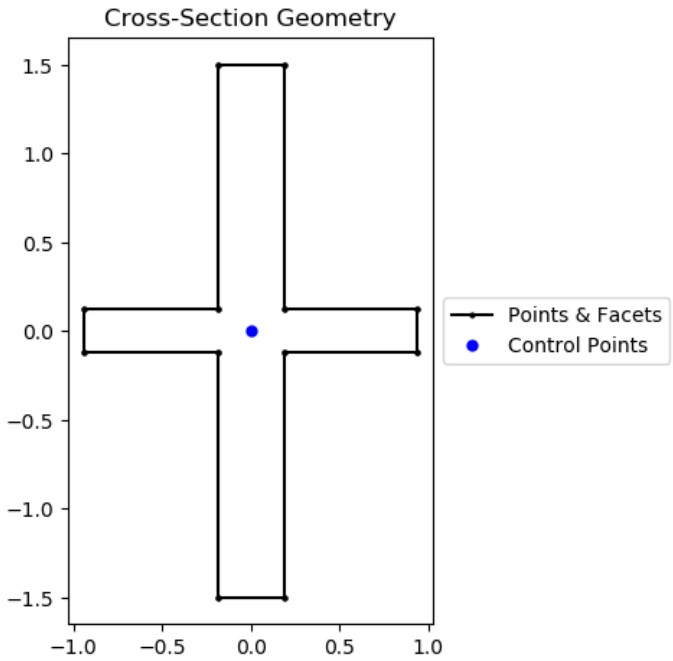


Fig. 51: Cruciform/cross section geometry.

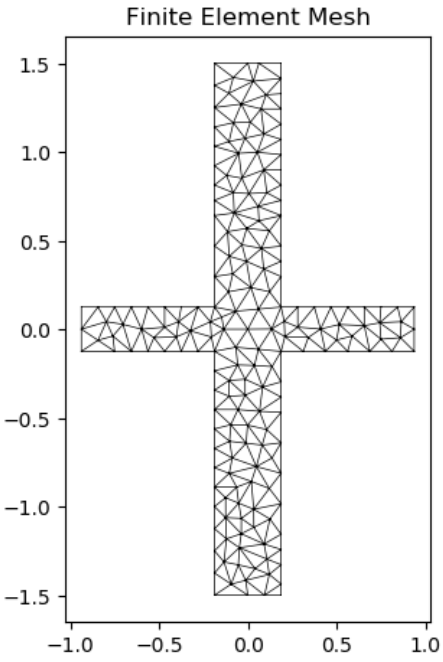


Fig. 52: Mesh generated from the above geometry.

DBOXSection Class

class sectionproperties.pre.nastran_sections.**DBOXSection**(*DIM1, DIM2, DIM3, DIM4, DIM5, DIM6, DIM7, DIM8, DIM9, DIM10, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a DBOX section with the center at the origin *(0, 0)*, with ten parameters defining dimensions. See MSC Nastran documentation[1] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Width (x) of the DBOX-section
- **DIM2** (*float*) – Depth (y) of the DBOX-section
- **DIM3** (*float*) – Width (x) of left-side box
- **DIM4** (*float*) – Thickness of left wall
- **DIM5** (*float*) – Thickness of center wall
- **DIM6** (*float*) – Thickness of right wall
- **DIM7** (*float*) – Thickness of top left wall
- **DIM8** (*float*) – Thickness of bottom left wall
- **DIM9** (*float*) – Thickness of top right wall
- **DIM10** (*float*) – Thickness of bottom right wall
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a DBOX cross-section with a depth of 3.0 and width of 8.0, and generates a mesh with a maximum triangular area of 0.01:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.DBOXSection(
    DIM1=8.0, DIM2=3.0, DIM3=3.0, DIM4=0.5, DIM5=0.625, DIM6=0.75, DIM7=0.375,␣
↪DIM8=0.25,
    DIM9=0.5, DIM10=0.375
)
mesh = geometry.create_mesh(mesh_sizes=[0.01])
```

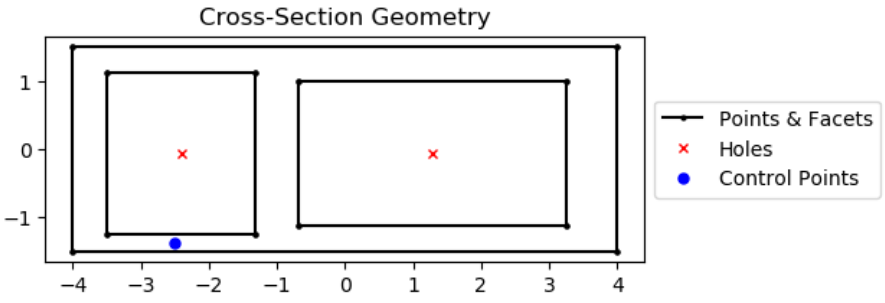


Fig. 53: DBOX section geometry.

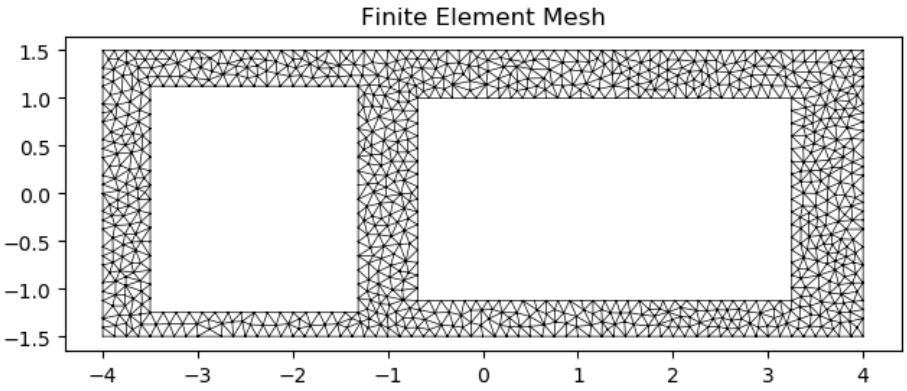


Fig. 54: Mesh generated from the above geometry.

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

FCROSSSection Class

class sectionproperties.pre.nastran_sections.**FCROSSSection**(*DIM1, DIM2, DIM3, DIM4, DIM5, DIM6, DIM7, DIM8, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a flanged cruciform/cross section with the intersection's middle center at the origin *(0, 0)*, with eight parameters defining dimensions. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Depth (y) of flanged cruciform
- **DIM2** (*float*) – Width (x) of flanged cruciform
- **DIM3** (*float*) – Thickness of vertical web
- **DIM4** (*float*) – Thickness of horizontal web
- **DIM5** (*float*) – Length of flange attached to vertical web
- **DIM6** (*float*) – Thickness of flange attached to vertical web
- **DIM7** (*float*) – Length of flange attached to horizontal web
- **DIM8** (*float*) – Thickness of flange attached to horizontal web
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example demonstrates the creation of a flanged cross section:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.FCROSSSection(
    DIM1=9.0, DIM2=6.0, DIM3=0.75, DIM4=0.625, DIM5=2.1, DIM6=0.375, DIM7=4.5,␣
↪DIM8=0.564
)
mesh = geometry.create_mesh(mesh_sizes=[0.03])
```

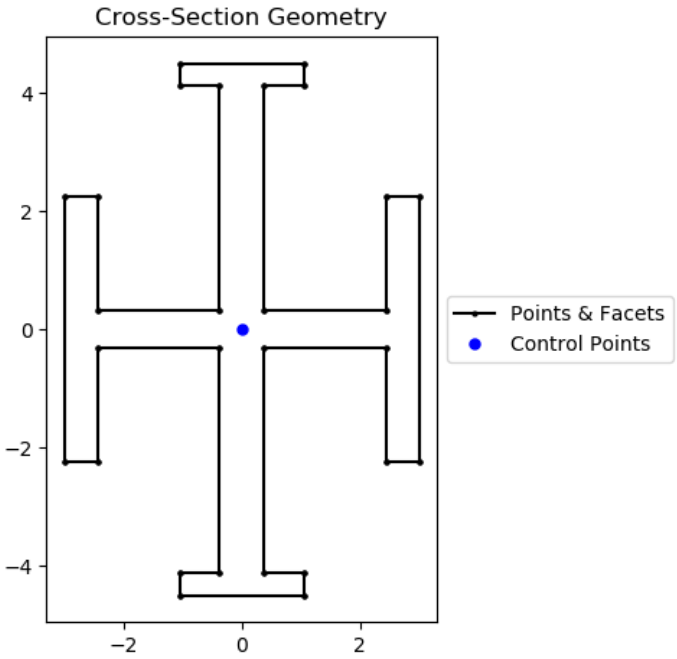


Fig. 55: Flanged Cruciform/cross section geometry.

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

GBOXSection Class

class sectionproperties.pre.nastran_sections.**GBOXSection**(*DIM1, DIM2, DIM3, DIM4, DIM5, DIM6, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a GBOX section with the center at the origin *(0, 0)*, with six parameters defining dimensions. See ASTROS documentation[5] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Width (x) of the GBOX-section
- **DIM2** (*float*) – Depth (y) of the GBOX-section
- **DIM3** (*float*) – Thickness of top flange

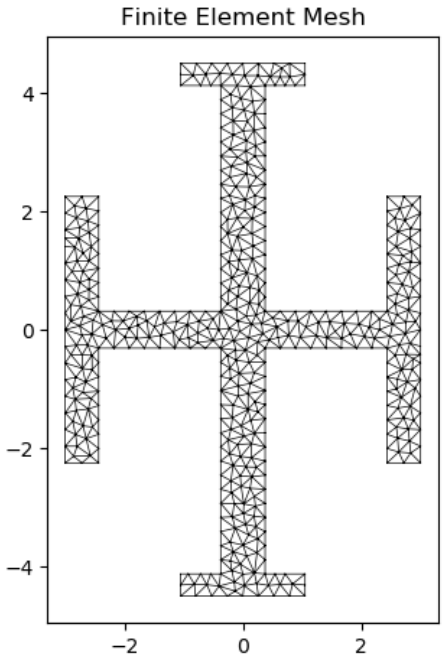


Fig. 56: Mesh generated from the above geometry.

- **DIM4** (*float*) – Thickness of bottom flange
- **DIM5** (*float*) – Thickness of webs
- **DIM6** (*float*) – Spacing between webs
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a GBOX cross-section with a depth of 2.5 and width of 6.0, and generates a mesh with a maximum triangular area of 0.01:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.GBOXSection(
    DIM1=6.0, DIM2=2.5, DIM3=0.375, DIM4=0.25, DIM5=0.625, DIM6=1.0
)
mesh = geometry.create_mesh(mesh_sizes=[0.01])
```

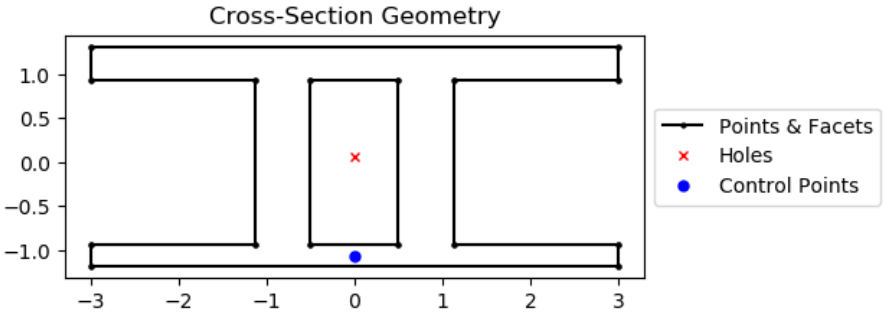


Fig. 57: GBOX section geometry.

getStressPoints (*shift=(0.0, 0.0)*)

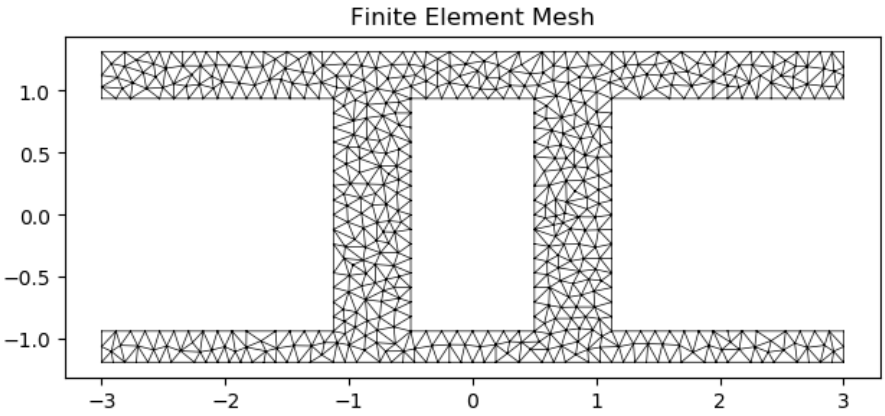


Fig. 58: Mesh generated from the above geometry.

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

HSection Class

class sectionproperties.pre.nastran_sections.**HSection**(*DIM1, DIM2, DIM3, DIM4, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a H section with the middle web's middle center at the origin *(0, 0)*, with four parameters defining dimensions. See Nastran documentation[1234] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Spacing between vertical flanges (length of web)
- **DIM2** (*float*) – Twice the thickness of the vertical flanges
- **DIM3** (*float*) – Depth (y) of the H-section
- **DIM4** (*float*) – Thickness of the middle web
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a H cross-section with a depth of 3.5 and width of 2.75, and generates a mesh with a maximum triangular area of 0.005:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.HSection(DIM1=2.0, DIM2=0.75, DIM3=3.5, DIM4=0.25)
mesh = geometry.create_mesh(mesh_sizes=[0.005])
```

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

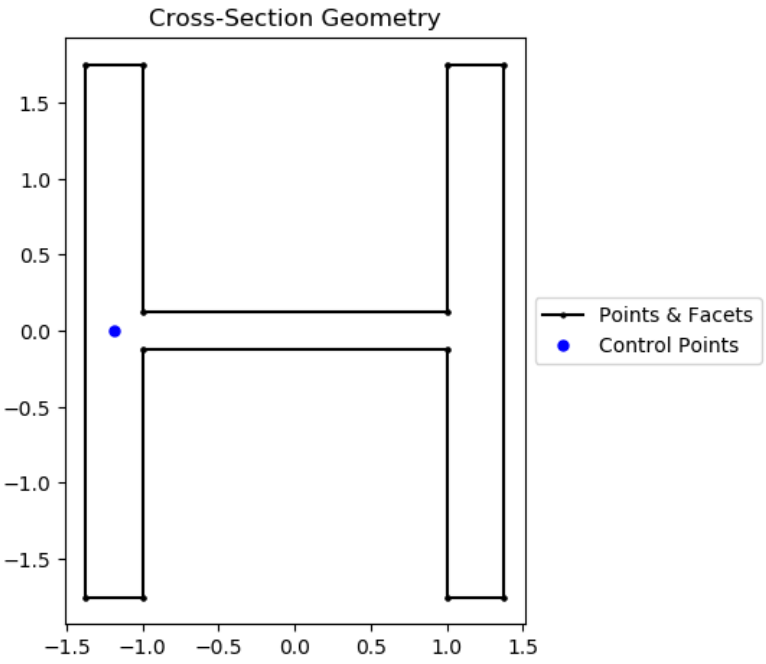


Fig. 59: H section geometry.

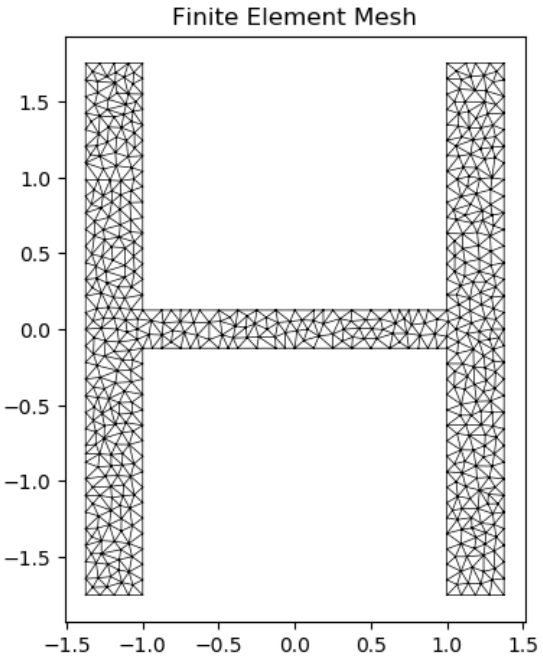


Fig. 60: Mesh generated from the above geometry.

Returns Stress evaluation points relative to shifted origin - C, D, E, F

HATSection Class

class sectionproperties.pre.nastran_sections.**HATSection**(*DIM1, DIM2, DIM3, DIM4, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a Hat section with the top most section's middle center at the origin *(0, 0)*, with four parameters defining dimensions. See Nastran documentation[1234] for more details. Note that HAT in ASTROS is actually HAT1 in this code. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Depth (y) of HAT-section
- **DIM2** (*float*) – Thickness of HAT-section
- **DIM3** (*float*) – Width (x) of top most section
- **DIM4** (*float*) – Width (x) of bottom sections
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a HAT cross-section with a depth of 1.25 and width of 2.5, and generates a mesh with a maximum triangular area of 0.001:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.HATSection(DIM1=1.25, DIM2=0.25, DIM3=1.5, DIM4=0.5)
mesh = geometry.create_mesh(mesh_sizes=[0.001])
```

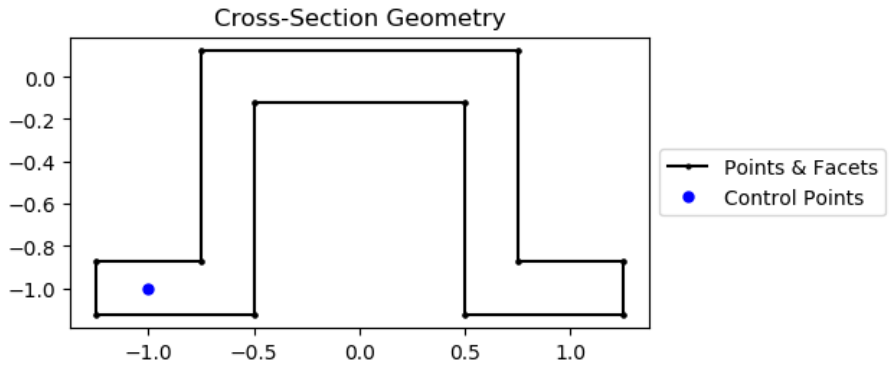


Fig. 61: HAT section geometry.

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the origin by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

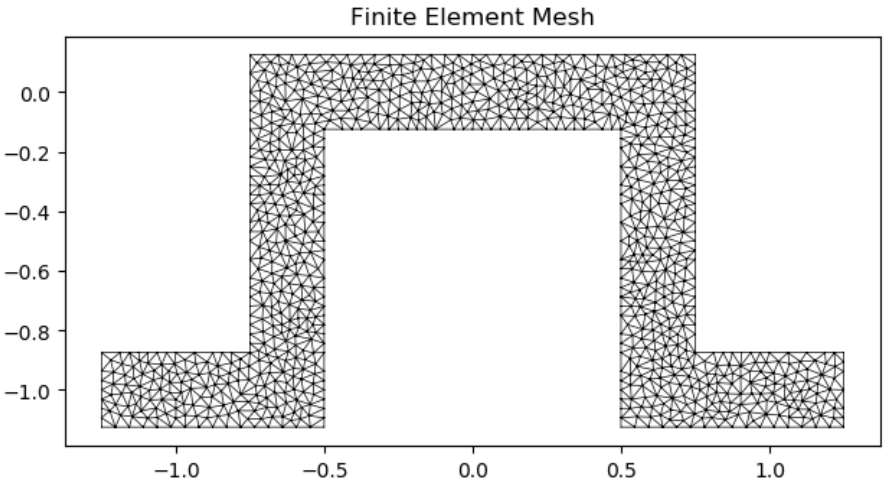


Fig. 62: Mesh generated from the above geometry.

HAT1Section Class

class sectionproperties.pre.nastran_sections.**HAT1Section**(*DIM1, DIM2, DIM3, DIM4, DIM5, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a HAT1 section with the bottom plate's bottom center at the origin *(0, 0)*, with five parameters defining dimensions. See Nastran documentation[1235] for definition of parameters. Note that in ASTROS, HAT1 is called HAT. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Width(x) of the HAT1-section
- **DIM2** (*float*) – Depth (y) of the HAT1-section
- **DIM3** (*float*) – Width (x) of hat's top flange
- **DIM4** (*float*) – Thickness of hat stiffener
- **DIM5** (*float*) – Thicknesss of bottom plate
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a HAT1 cross-section with a depth of 2.0 and width of 4.0, and generates a mesh with a maximum triangular area of 0.005:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.HAT1Section(DIM1=4.0, DIM2=2.0, DIM3=1.5, DIM4=0.1875,␣
↪DIM5=0.375)
mesh = geometry.create_mesh(mesh_sizes=[0.005])
```

create_mesh(*mesh_sizes*)

Creates a quadratic triangular mesh from the Geometry object. This is overloaded here to allow specifying only one mesh_size which is used for both regions in the Hat1 section.

Parameters **mesh_sizes** – A list of maximum element areas corresponding to each region within the cross-section geometry.

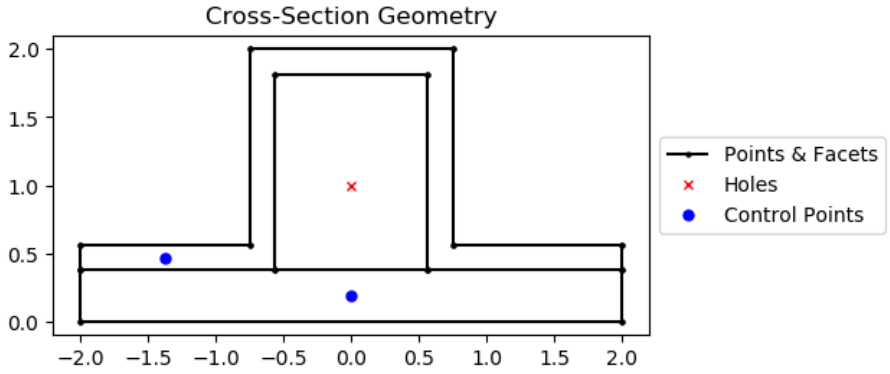


Fig. 63: HAT1 section geometry.

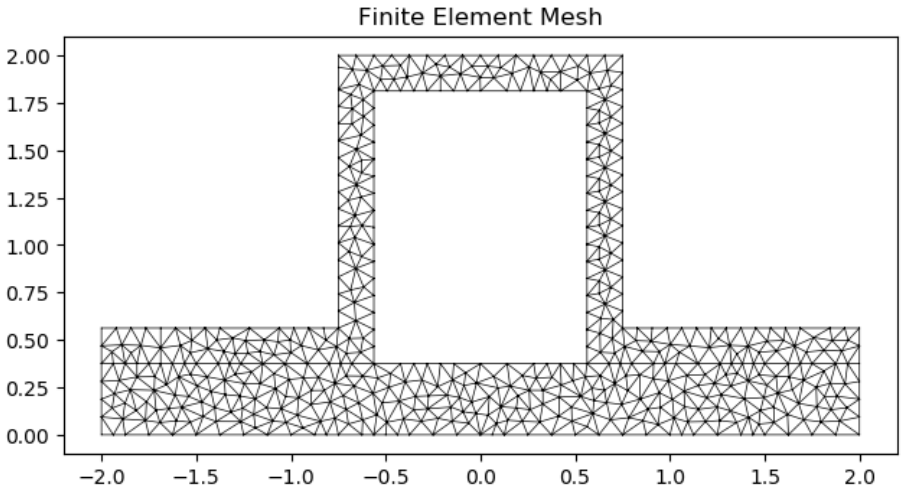


Fig. 64: Mesh generated from the above geometry.

Returns Object containing generated mesh data

Return type `meshpy.triangle.MeshInfo`

Raises **`AssertionError`** – If the number of mesh sizes does not match the number of regions

`getStressPoints`(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **`shift`** (*`tuple(float, float)`*) – Vector that shifts the origin by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

HEXASection Class

`class` `sectionproperties.pre.nastran_sections.`**`HEXASection`**(*DIM1, DIM2, DIM3, shift=[0, 0]*)

Bases: *`sectionproperties.pre.sections.Geometry`*

Constructs a HEXA (hexagon) section with the center at the origin *(0, 0)*, with three parameters defining dimensions. See Nastran documentation[1234] for more details. Added by JohnDN90.

Parameters

- **`DIM1`** (*`float`*) – Spacing between bottom right point and right most point
- **`DIM2`** (*`float`*) – Width (x) of hexagon
- **`DIM3`** (*`float`*) – Depth (y) of hexagon
- **`shift`** (*`list[float, float]`*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a rectangular cross-section with a depth of 1.5 and width of 2.0, and generates a mesh with a maximum triangular area of 0.005:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.HEXASection(DIM1=0.5, DIM2=2.0, DIM3=1.5)
mesh = geometry.create_mesh(mesh_sizes=[0.005])
```

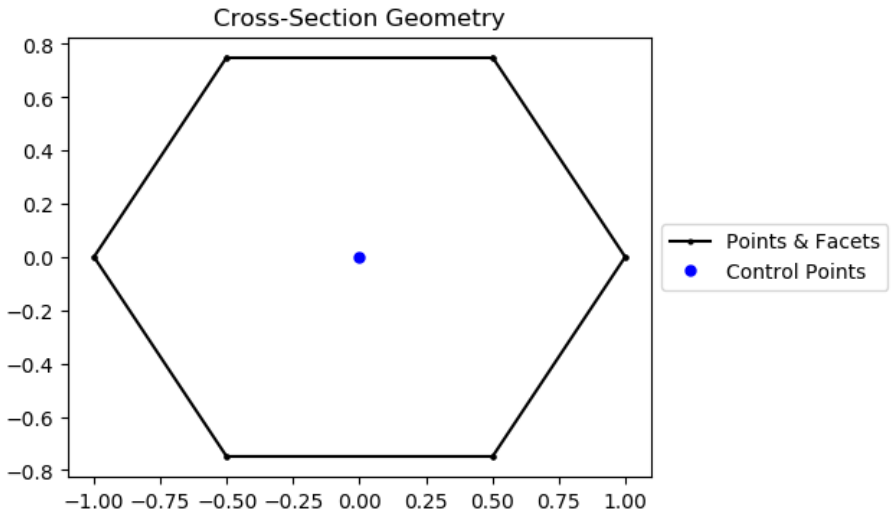


Fig. 65: HEXA section geometry.

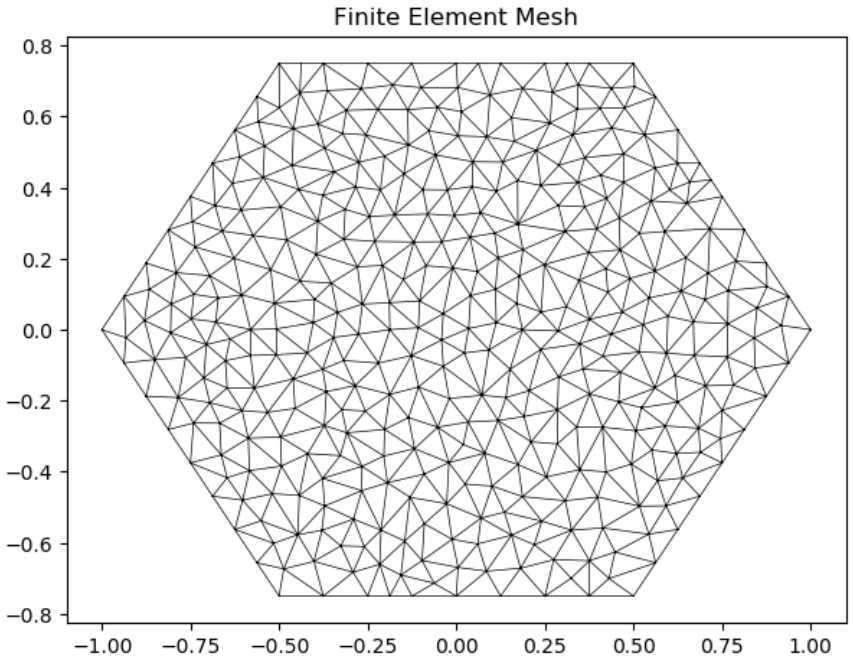


Fig. 66: Mesh generated from the above geometry.

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

NISection Class

class sectionproperties.pre.nastran_sections.**NISection**(*DIM1, DIM2, DIM3, DIM4, DIM5, DIM6, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs Nastran's I section with the bottom flange's middle center at the origin *(0, 0)*, with six parameters defining dimensions. See Nastran documentation[1234] for definition of parameters. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Depth(y) of the I-section
- **DIM2** (*float*) – Width (x) of bottom flange
- **DIM3** (*float*) – Width (x) of top flange
- **DIM4** (*float*) – Thickness of web
- **DIM5** (*float*) – Thickness of bottom web
- **DIM6** (*float*) – Thickness of top web
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a Nastran I cross-section with a depth of 5.0, and generates a mesh with a maximum triangular area of 0.008:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.NISection(
    DIM1=5.0, DIM2=2.0, DIM3=3.0, DIM4=0.25, DIM5=0.375, DIM6=0.5
)
mesh = geometry.create_mesh(mesh_sizes=[0.008])
```

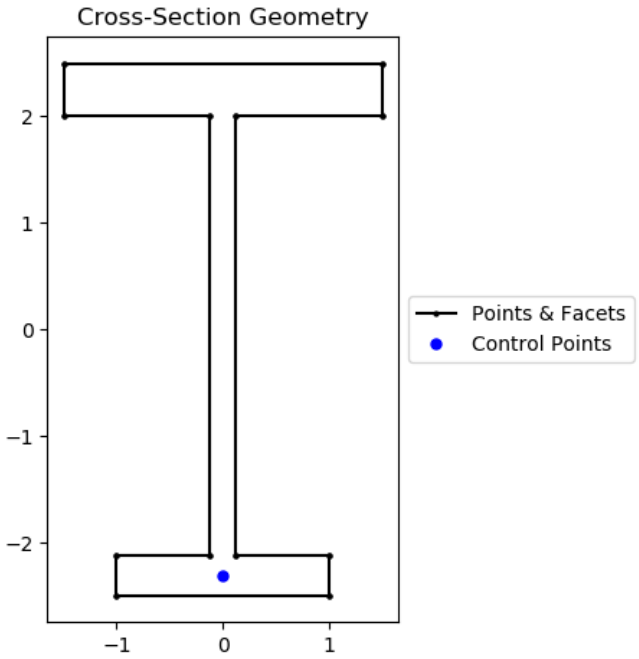


Fig. 67: Nastran's I section geometry.

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

I1Section Class

class sectionproperties.pre.nastran_sections.**I1Section**(*DIM1, DIM2, DIM3, DIM4, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a I1 section with the web's middle center at the origin *(0, 0)*, with four parameters defining dimensions. See Nastran documentation[1234] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Twice distance from web end to flange end
- **DIM2** (*float*) – Thickness of web

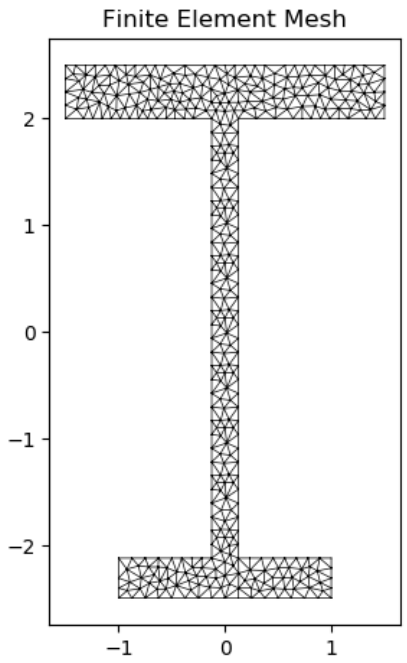


Fig. 68: Mesh generated from the above geometry.

- **DIM3** (*float*) – Length of web (spacing between flanges)
- **DIM4** (*float*) – Depth (y) of the I1-section
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a I1 cross-section with a depth of 5.0 and width of 1.75, and generates a mesh with a maximum triangular area of 0.02:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.I1Section(DIM1=1.0, DIM2=0.75, DIM3=4.0, DIM4=5.0)
mesh = geometry.create_mesh(mesh_sizes=[0.02])
```

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

LSection Class

class sectionproperties.pre.nastran_sections.**LSection**(*DIM1, DIM2, DIM3, DIM4, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a L section with the intersection's center at the origin *(0, 0)*, with four parameters defining dimensions. See Nastran documentation[123] for more details. Added by JohnDN90.

Parameters

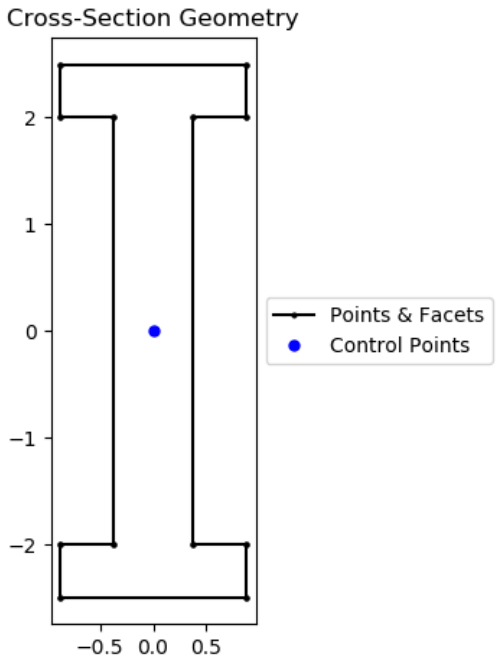


Fig. 69: I1 section geometry.

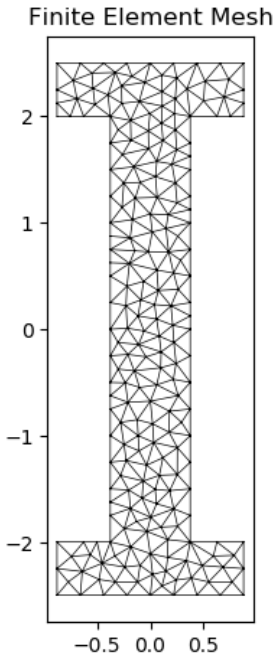


Fig. 70: Mesh generated from the above geometry.

- **DIM1** (*float*) – Width (x) of the L-section
- **DIM2** (*float*) – Depth (y) of the L-section
- **DIM3** (*float*) – Thickness of flange (horizontal portion)
- **DIM4** (*float*) – Thickness of web (vertical portion)
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a L cross-section with a depth of 6.0 and width of 3.0, and generates a mesh with a maximum triangular area of 0.01:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.LSection(DIM1=3.0, DIM2=6.0, DIM3=0.375, DIM4=0.625)
mesh = geometry.create_mesh(mesh_sizes=[0.01])
```

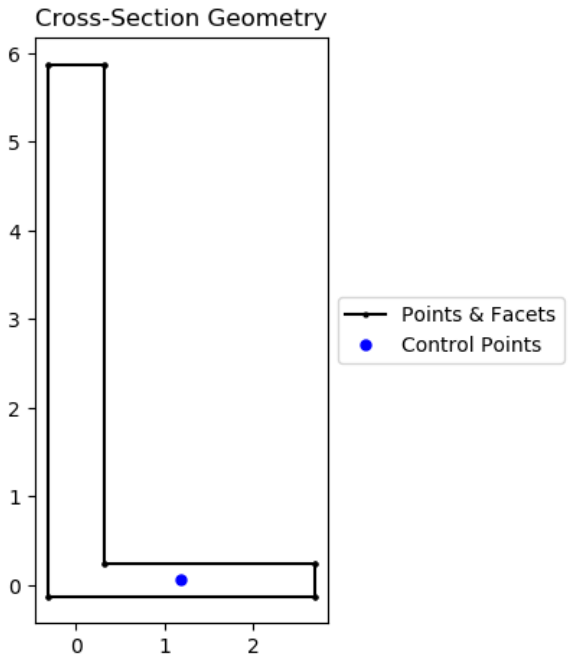


Fig. 71: L section geometry.

getStressPoints (*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

RODSection Class

class sectionproperties.pre.nastran_sections.**RODSection**(*DIM1, n, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a circular rod section with the center at the origin *(0, 0)*, with one parameter defining dimensions. See Nastran documentation[1234] for more details. Added by JohnDN90.

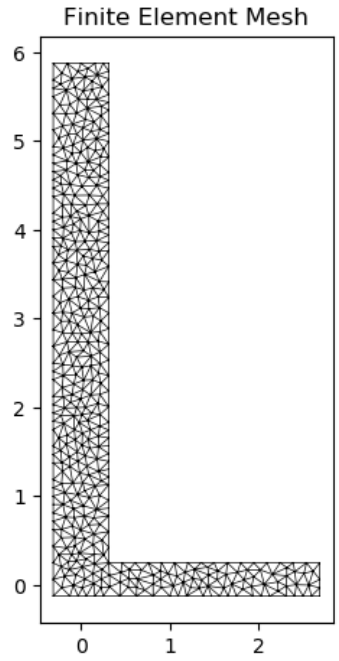


Fig. 72: Mesh generated from the above geometry.

Parameters

- **DIM1** (*float*) – Radius of the circular rod section
- **n** (*int*) – Number of points discretising the circle
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a circular rod with a radius of 3.0 and 50 points discretizing the boundary, and generates a mesh with a maximum triangular area of 0.01:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.RODSection(DIM1=3.0, n=50)
mesh = geometry.create_mesh(mesh_sizes=[0.01])
```

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters

- **DIM1** (*float*) – Radius of the circular rod section
- **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

TSection Class

class sectionproperties.pre.nastran_sections.**TSection**(*DIM1, DIM2, DIM3, DIM4, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

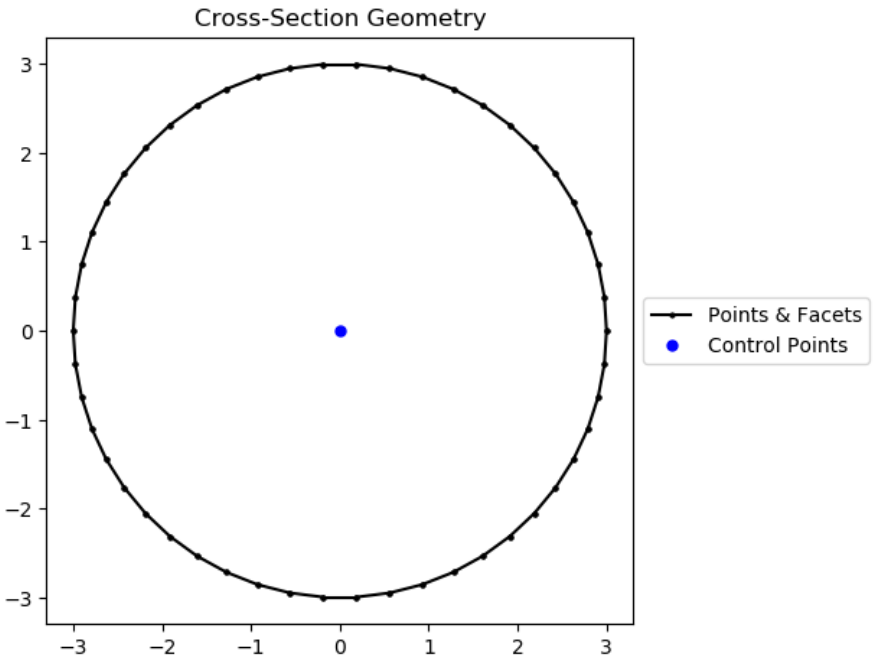


Fig. 73: Rod section geometry.

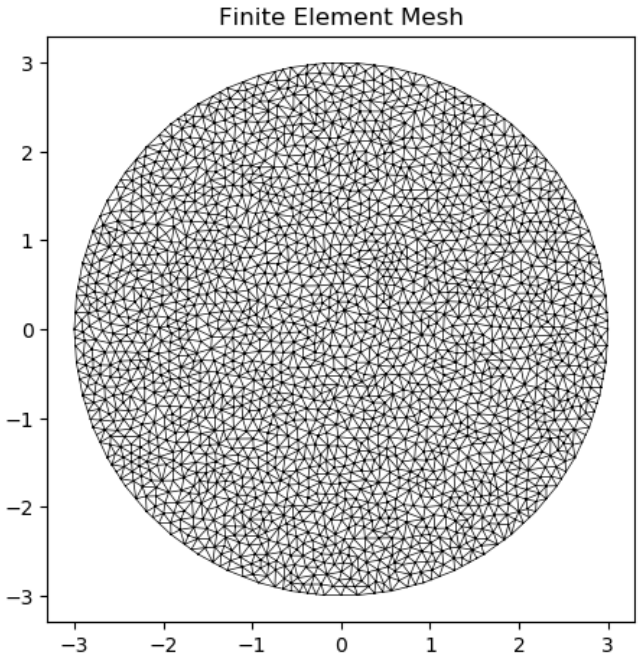


Fig. 74: Mesh generated from the above geometry.

Constructs a T section with the top flange's middle center at the origin *(0, 0)*, with four parameters defining dimensions. See Nastran documentation[12345] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Width (x) of top flange
- **DIM2** (*float*) – Depth (y) of the T-section
- **DIM3** (*float*) – Thickness of top flange
- **DIM4** (*float*) – Thickness of web
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a T cross-section with a depth of 4.0 and width of 3.0, and generates a mesh with a maximum triangular area of 0.001:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.TSection(DIM1=3.0, DIM2=4.0, DIM3=0.375, DIM4=0.25)
mesh = geometry.create_mesh(mesh_sizes=[0.001])
```

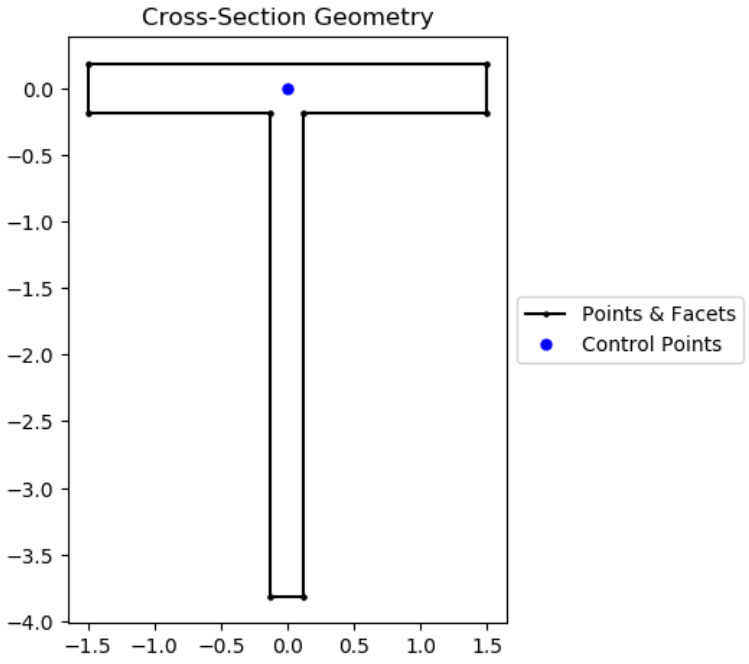


Fig. 75: T section geometry.

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

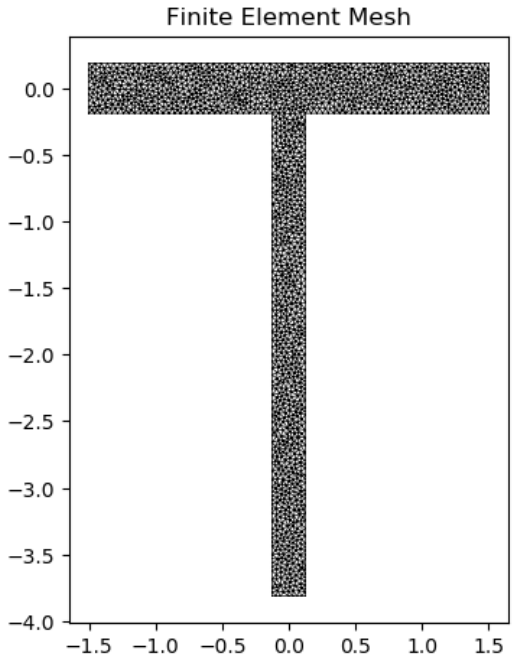


Fig. 76: Mesh generated from the above geometry.

T1Section Class

class sectionproperties.pre.nastran_sections.**T1Section**(*DIM1, DIM2, DIM3, DIM4, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a T1 section with the right flange's middle center at the origin *(0, 0)*, with four parameters defining dimensions. See Nastran documentation[1234] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Depth (y) of T1-section
- **DIM2** (*float*) – Length (x) of web
- **DIM3** (*float*) – Thickness of right flange
- **DIM4** (*float*) – Thickness of web
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a T1 cross-section with a depth of 3.0 and width of 3.875, and generates a mesh with a maximum triangular area of 0.001:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.T1Section(DIM1=3.0, DIM2=3.5, DIM3=0.375, DIM4=0.25)
mesh = geometry.create_mesh(mesh_sizes=[0.001])
```

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

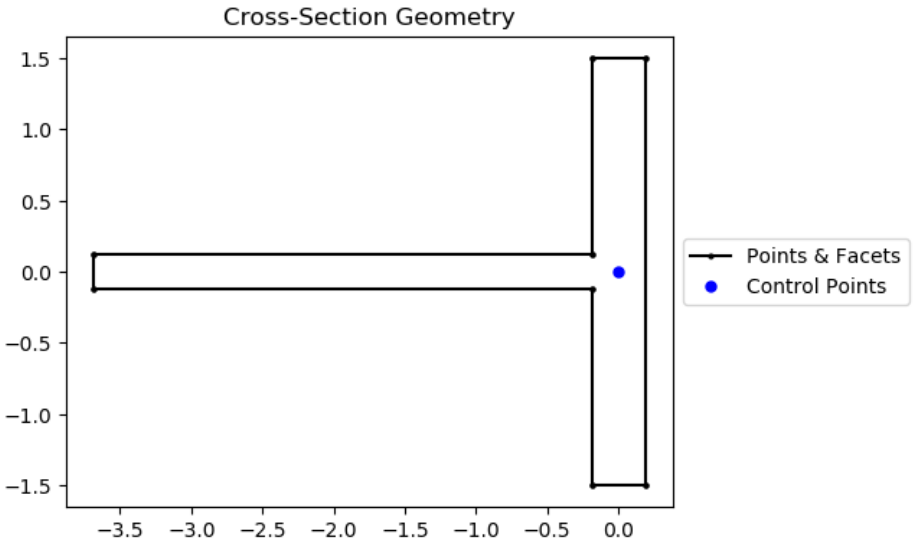


Fig. 77: T1 section geometry.

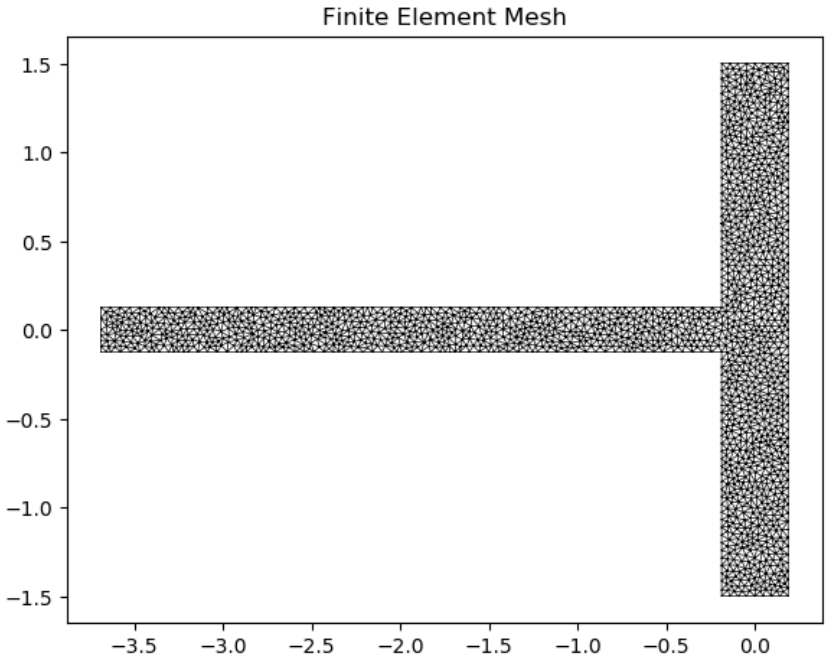


Fig. 78: Mesh generated from the above geometry.

Returns Stress evaluation points relative to shifted origin - C, D, E, F

T2Section Class

class sectionproperties.pre.nastran_sections.**T2Section**(*DIM1, DIM2, DIM3, DIM4, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a T2 section with the bottom flange's middle center at the origin *(0, 0)*, with four parameters defining dimensions. See Nastran documentation[1234] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Width (x) of T2-section
- **DIM2** (*float*) – Depth (y) of T2-section
- **DIM3** (*float*) – Thickness of bottom flange
- **DIM4** (*float*) – Thickness of web
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a T2 cross-section with a depth of 4.0 and width of 3.0, and generates a mesh with a maximum triangular area of 0.005:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.T2Section(DIM1=3.0, DIM2=4.0, DIM3=0.375, DIM4=0.5)
mesh = geometry.create_mesh(mesh_sizes=[0.005])
```

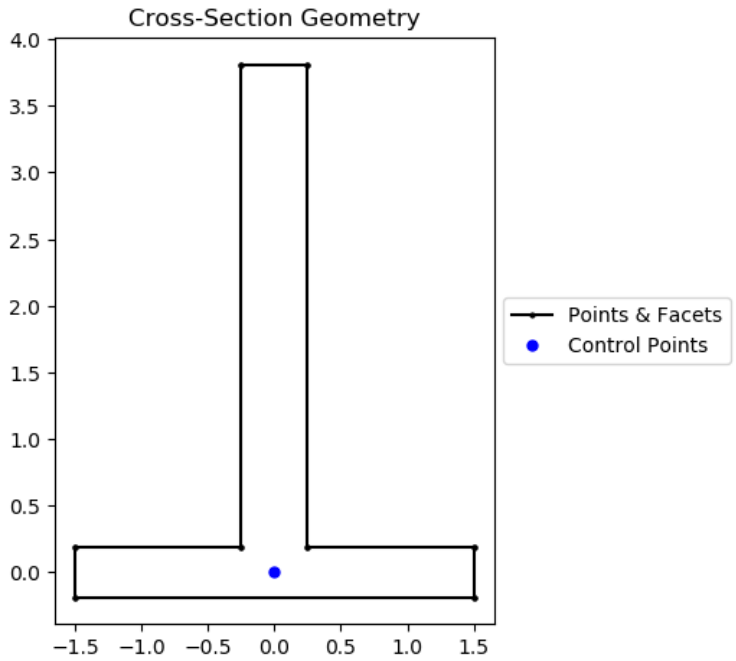


Fig. 79: T2 section geometry.

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

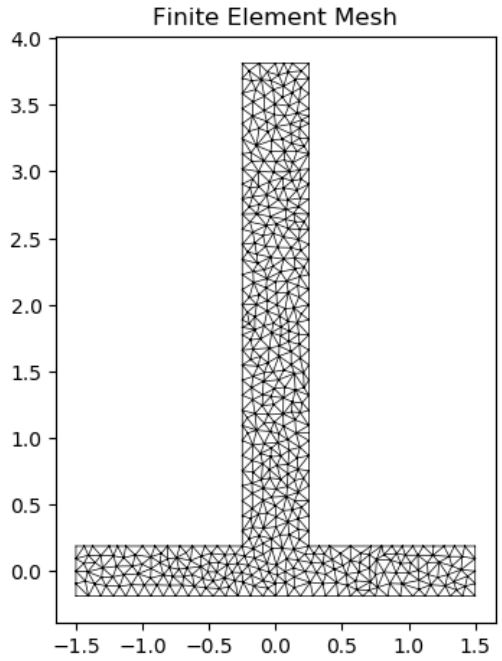


Fig. 80: Mesh generated from the above geometry.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

TUBESection Class

class sectionproperties.pre.nastran_sections.**TUBESection**(*DIM1, DIM2, n, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a circular tube section with the center at the origin *(0, 0)*, with two parameters defining dimensions. See Nastran documentation[1234] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Outer radius of the circular tube section
- **DIM2** (*float*) – Inner radius of the circular tube section
- **n** (*int*) – Number of points discretising the circle
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a circular tube cross-section with an outer radius of 3.0 and an inner radius of 2.5, and generates a mesh with 37 points discretizing the boundaries and a maximum triangular area of 0.01:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.TUBESection(DIM1=3.0, DIM2=2.5, n=37)
mesh = geometry.create_mesh(mesh_sizes=[0.01])
```

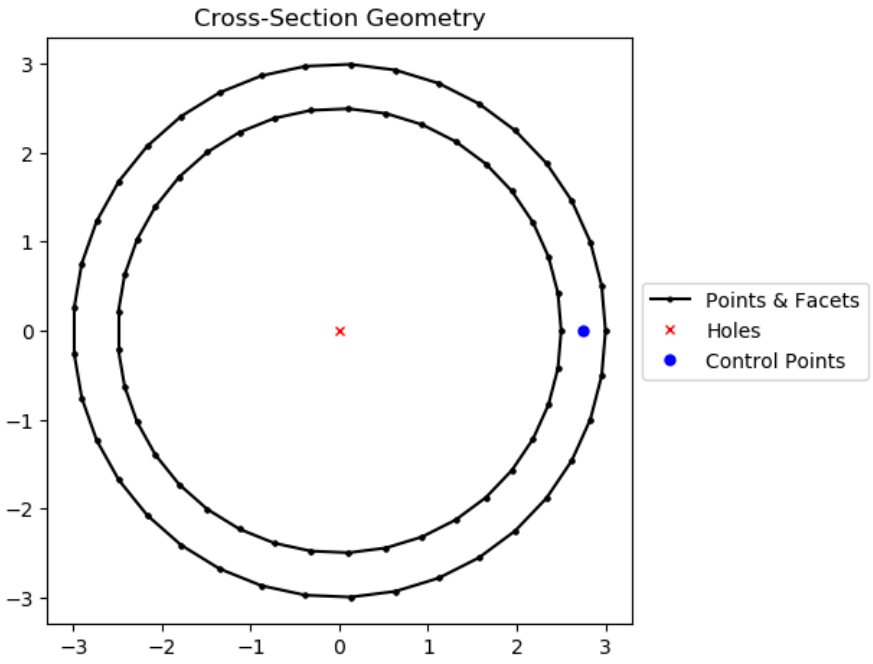


Fig. 81: TUBE section geometry.

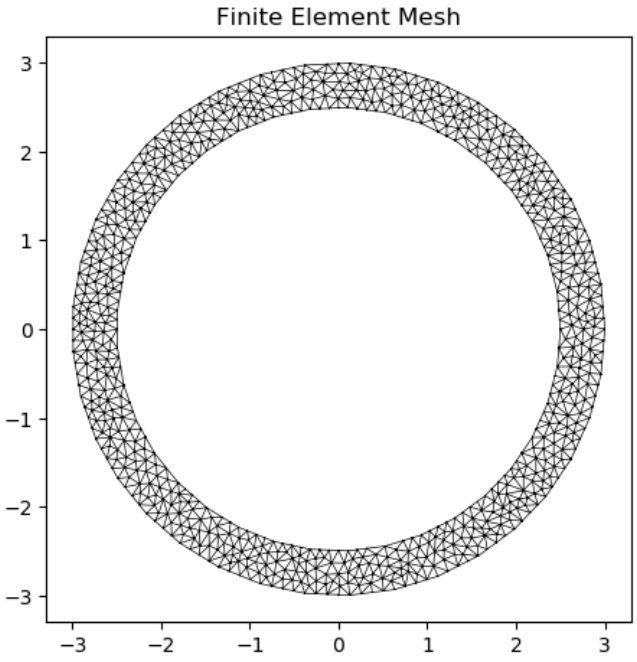


Fig. 82: Mesh generated from the above geometry.

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

TUBE2Section Class

class sectionproperties.pre.nastran_sections.**TUBE2Section**(*DIM1, DIM2, n, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a circular TUBE2 section with the center at the origin *(0, 0)*, with two parameters defining dimensions. See MSC Nastran documentation[1] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Outer radius of the circular tube section
- **DIM2** (*float*) – Thickness of wall
- **n** (*int*) – Number of points discretising the circle
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a ciruclar TUBE2 cross-section with an outer radius of 3.0 and a wall thickness of 0.5, and generates a mesh with 37 point discritizing the boundary and a maximum triangular area of 0.01:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.TUBE2Section(DIM1=3.0, DIM2=0.5, n=37)
mesh = geometry.create_mesh(mesh_sizes=[0.01])
```

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

ZSection Class

class sectionproperties.pre.nastran_sections.**ZSection**(*DIM1, DIM2, DIM3, DIM4, shift=[0, 0]*)

Bases: *sectionproperties.pre.sections.Geometry*

Constructs a Z section with the web's middle center at the origin *(0, 0)*, with four parameters defining dimensions. See Nastran documentation[1234] for more details. Added by JohnDN90.

Parameters

- **DIM1** (*float*) – Width (x) of horizontal members
- **DIM2** (*float*) – Thickness of web
- **DIM3** (*float*) – Spacing between horizontal members (length of web)

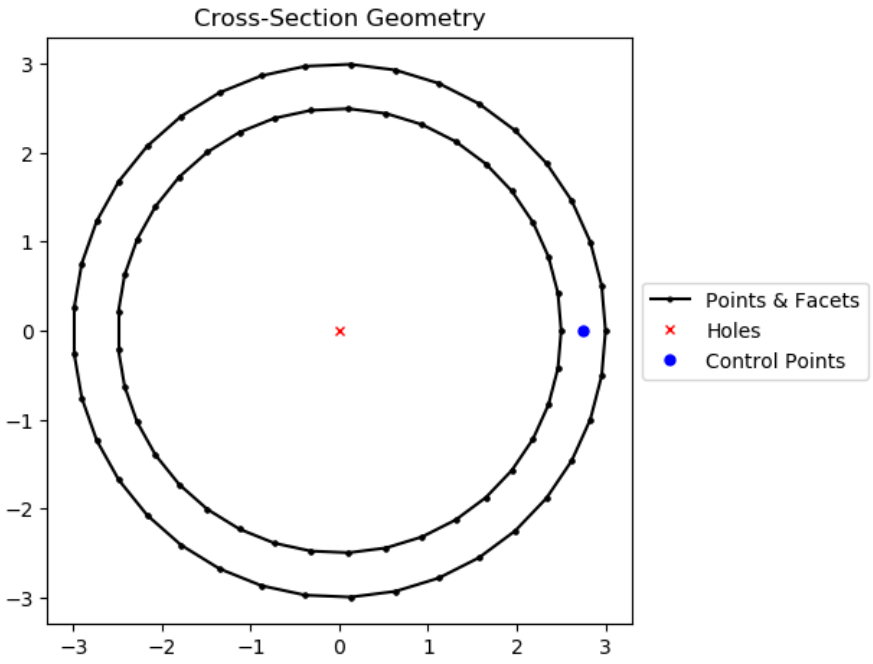


Fig. 83: TUBE2 section geometry.

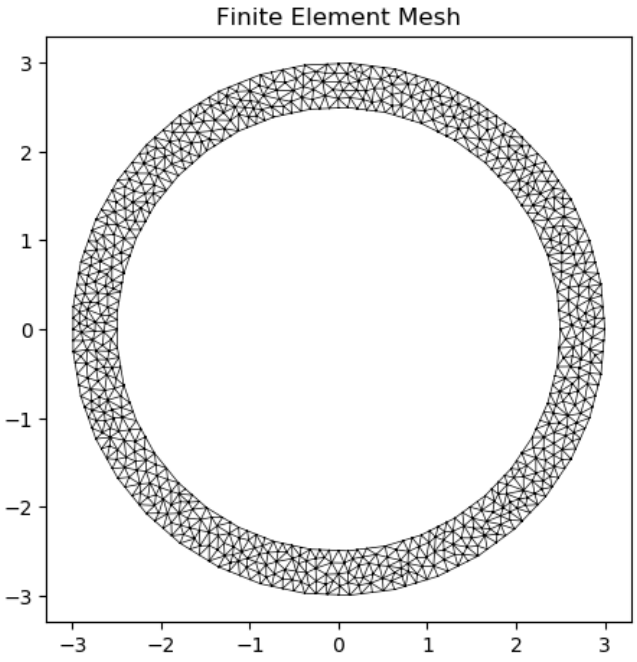


Fig. 84: Mesh generated from the above geometry.

- **DIM4** (*float*) – Depth (y) of Z-section
- **shift** (*list[float, float]*) – Vector that shifts the cross-section by *(x, y)*

The following example creates a rectangular cross-section with a depth of 4.0 and width of 2.75, and generates a mesh with a maximum triangular area of 0.005:

```
import sectionproperties.pre.nastran_sections as nsections

geometry = nsections.ZSection(DIM1=1.125, DIM2=0.5, DIM3=3.5, DIM4=4.0)
mesh = geometry.create_mesh(mesh_sizes=[0.005])
```

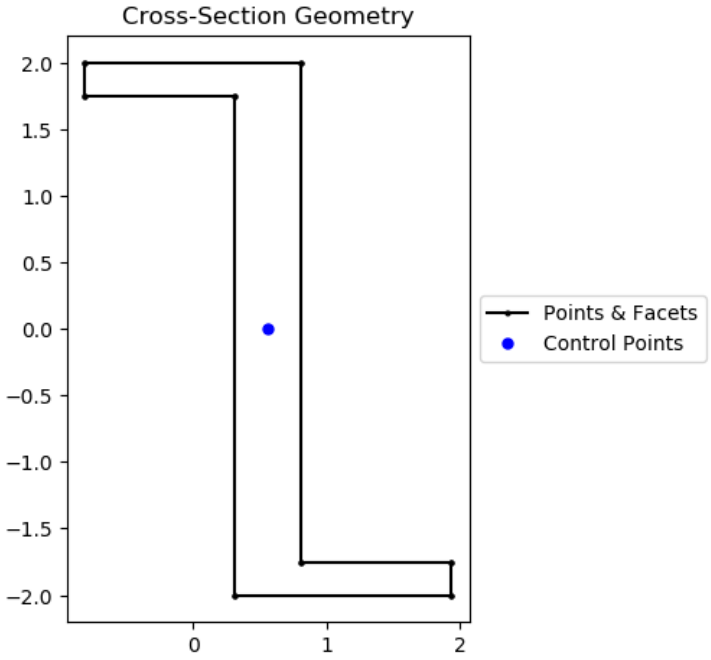


Fig. 85: Z section geometry.

getStressPoints(*shift=(0.0, 0.0)*)

Returns the coordinates of the stress evaluation points relative to the origin of the cross-section. The shift parameter can be used to make the coordinates relative to the centroid or the shear center.

Parameters **shift** (*tuple(float, float)*) – Vector that shifts the cross-section by *(x, y)*

Returns Stress evaluation points relative to shifted origin - C, D, E, F

References

7.2 Analysis Package

7.2.1 *cross_section* Module

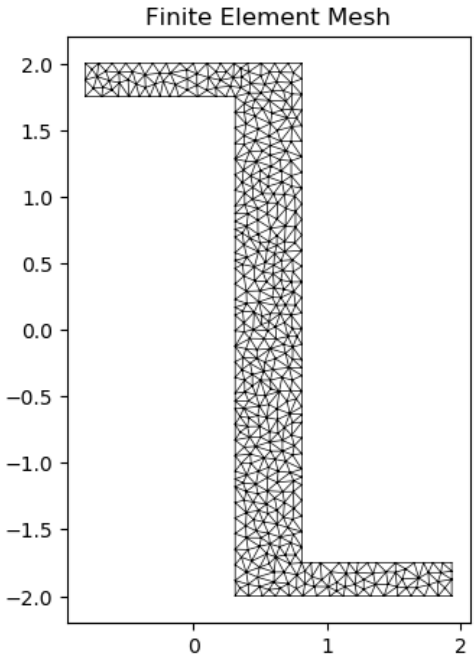


Fig. 86: Mesh generated from the above geometry.

CrossSection Class

class sectionproperties.analysis.cross_section.**CrossSection**(*geometry*, *mesh*, *materials=None*, *time_info=False*)

Bases: `object`

Class for structural cross-sections.

Stores the finite element geometry, mesh and material information and provides methods to compute the cross-section properties. The element type used in this program is the six-noded quadratic triangular element.

The constructor extracts information from the provided mesh object and creates and stores the corresponding Tri6 finite element objects.

Parameters

- **geometry** (*Geometry*) – Cross-section geometry object used to generate the mesh
- **mesh** (`meshpy.triangle.MeshInfo`) – Mesh object returned by meshpy
- **materials** (list[*Material*]) – A list of material properties corresponding to various regions in the geometry and mesh. Note that if materials are specified, the number of material objects ust equal the number of regions in the geometry. If no materials are specified, only a purely geometric analysis can take place, and all regions will be assigned a default material with an elastic modulus and yield strength equal to 1, and a Poisson's ratio equal to 0.
- **time_info** (*bool*) – If set to True, a detailed description of the computation and

the time cost is printed to the terminal.

The following example creates a *CrossSection* object of a 100D x 50W rectangle using a mesh size of 5:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.RectangularSection(d=100, b=50)
mesh = geometry.create_mesh(mesh_sizes=[5])
section = CrossSection(geometry, mesh)
```

The following example creates a 100D x 50W rectangle, with the top half of the section comprised of timber and the bottom half steel. The timber section is meshed with a maximum area of 10 and the steel section mesh with a maximum area of 5:

```
import sectionproperties.pre.sections as sections
from sectionproperties.pre.pre import Material
from sectionproperties.analysis.cross_section import CrossSection

geom_steel = sections.RectangularSection(d=50, b=50)
geom_timber = sections.RectangularSection(d=50, b=50, shift=[0, 50])
geometry = sections.MergedSection([geom_steel, geom_timber])
geometry.clean_geometry()

mesh = geometry.create_mesh(mesh_sizes=[5, 10])

steel = Material(
    name='Steel', elastic_modulus=200e3, poissons_ratio=0.3, yield_
↪strength=250,
    color='grey'
)
timber = Material(
    name='Timber', elastic_modulus=8e3, poissons_ratio=0.35, yield_
↪strength=20,
    color='burlywood'
)

section = CrossSection(geometry, mesh, [steel, timber])
section.plot_mesh(materials=True, alpha=0.5)
```

Variables

- **elements** (list[*Tri6*]) – List of finite element objects describing the cross-section mesh
- **num_nodes** (*int*) – Number of nodes in the finite element mesh
- **geometry** (*Geometry*) – Cross-section geometry object used to generate the mesh
- **mesh** (meshpy.triangle.MeshInfo) – Mesh object returned by meshpy
- **mesh_nodes** (numpy.ndarray) – Array of node coordinates from the mesh
- **mesh_elements** (numpy.ndarray) – Array of connectivities from the mesh
- **mesh_attributes** (numpy.ndarray) – Array of attributes from the mesh
- **materials** – List of materials

- **material_groups** – List of objects containing the elements in each defined material
- **section_props** (*SectionProperties*) – Class to store calculated section properties

Raises **AssertionError** – If the number of materials does not equal the number of regions

assemble_torsion(*lg=True*)

Assembles stiffness matrices to be used for the computation of warping properties and the torsion load vector (f_torsion). Both a regular (k) and Lagrangian multiplier (k_lg) stiffness matrix are returned. The stiffness matrices are assembled using the sparse COO format and returned in the sparse CSC format.

Parameters **lg** (*bool*) – Whether or not to calculate the Lagrangian multiplier stiffness matrix

Returns Regular stiffness matrix, Lagrangian multiplier stiffness matrix and torsion load vector *(k, k_lg, f_torsion)*

Return type tuple(scipy.sparse.csc_matrix, scipy.sparse.csc_matrix, numpy.ndarray)

calculate_frame_properties(*time_info=False*, *solver_type='direct'*)

Calculates and returns the properties required for a frame analysis. The properties are also stored in the *SectionProperties* object contained in the section_props class variable.

Parameters

- **time_info** (*bool*) – If set to True, a detailed description of the computation and the time cost is printed to the terminal.
- **solver_type** (*string*) – Solver used for solving systems of linear equations, either using the *'direct'* method or *'cgs'* iterative method

Returns Cross-section properties to be used for a frame analysis *(area, ixx, iyy, ixy, j, phi)*

Return type tuple(float, float, float, float, float, float)

The following section properties are calculated:

- Cross-sectional area *(area)*
- Second moments of area about the centroidal axis *(ixx, iyy, ixy)*
- Torsion constant *(j)*
- Principal axis angle *(phi)*

If materials are specified for the cross-section, the area, second moments of area and torsion constant are elastic moulus weighted.

The following example demonstrates the use of this method:

```
section = CrossSection(geometry, mesh)
(area, ixx, iyy, ixy, j, phi) = section.calculate_frame_properties()
```

calculate_geometric_properties(*time_info=False*)

Calculates the geometric properties of the cross-section and stores them in the *SectionProperties* object contained in the section_props class variable.

Parameters **time_info** (*bool*) – If set to True, a detailed description of the computation and the time cost is printed to the terminal.

The following geometric section properties are calculated:

- Cross-sectional area
- Cross-sectional perimeter
- Modulus weighted area (axial rigidity)
- First moments of area

- Second moments of area about the global axis
- Second moments of area about the centroidal axis
- Elastic centroid
- Centroidal section moduli
- Radii of gyration
- Principal axis properties

If materials are specified for the cross-section, the moments of area and section moduli are elastic modulus weighted.

The following example demonstrates the use of this method:

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
```

calculate_plastic_properties(*time_info=False*, *verbose=False*, *debug=False*)

Calculates the plastic properties of the cross-section and stores the, in the *SectionProperties* object contained in the `section_props` class variable.

Parameters

- **time_info** (*bool*) – If set to True, a detailed description of the computation and the time cost is printed to the terminal.
- **verbose** (*bool*) – If set to True, the number of iterations required for each plastic axis is printed to the terminal.
- **debug** (*bool*) – If set to True, the geometry is plotted each time a new mesh is generated by the plastic centroid algorithm.

The following warping section properties are calculated:

- Plastic centroid for bending about the centroidal and principal axes
- Plastic section moduli for bending about the centroidal and principal axes
- Shape factors for bending about the centroidal and principal axes

If materials are specified for the cross-section, the plastic section moduli are displayed as plastic moments (i.e $M_p = f_y S$) and the shape factors are not calculated.

Note that the geometric properties must be calculated before the plastic properties are calculated:

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
```

Raises **RuntimeError** – If the geometric properties have not been calculated prior to calling this method

calculate_stress(*N=0*, *Vx=0*, *Vy=0*, *Mxx=0*, *Myy=0*, *M11=0*, *M22=0*, *Mzz=0*, *time_info=False*)

Calculates the cross-section stress resulting from design actions and returns a *StressPost* object allowing post-processing of the stress results.

Parameters

- **N** (*float*) – Axial force
- **Vx** (*float*) – Shear force acting in the x-direction
- **Vy** (*float*) – Shear force acting in the y-direction
- **Mxx** (*float*) – Bending moment about the centroidal xx-axis
- **Myy** (*float*) – Bending moment about the centroidal yy-axis
- **M11** (*float*) – Bending moment about the centroidal 11-axis
- **M22** (*float*) – Bending moment about the centroidal 22-axis
- **Mzz** (*float*) – Torsion moment about the centroidal zz-axis
- **time_info** (*bool*) – If set to True, a detailed description of the computation and the time cost is printed to the terminal.

Returns Object for post-processing cross-section stresses
Return type *StressPost*

Note that a geometric and warping analysis must be performed before a stress analysis is carried out:

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(N=1e3, Vy=3e3, Mxx=1e6)
```

Raises **RuntimeError** – If a geometric and warping analysis have not been performed prior to calling this method

calculate_warping_properties(*time_info=False*, *solver_type='direct'*)

Calculates all the warping properties of the cross-section and stores them in the *SectionProperties* object contained in the `section_props` class variable.

Parameters

- **time_info** (*bool*) – If set to True, a detailed description of the computation and the time cost is printed to the terminal.
- **solver_type** (*string*) – Solver used for solving systems of linear equations, either using the *'direct'* method or *'cgs'* iterative method

The following warping section properties are calculated:

- Torsion constant
- Shear centre
- Shear area
- Warping constant
- Monosymmetry constant

If materials are specified, the values calculated for the torsion constant, warping constant and shear area are elastic modulus weighted.

Note that the geometric properties must be calculated first for the calculation of the warping properties to be correct:

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
```

Raises **RuntimeError** – If the geometric properties have not been calculated prior to calling this method

display_mesh_info()

Prints mesh statistics (number of nodes, elements and regions) to the command window.

The following example displays the mesh statistics for a Tee section merged from two rectangles:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

rec1 = sections.RectangularSection(d=100, b=25, shift=[-12.5, 0])
rec2 = sections.RectangularSection(d=25, b=100, shift=[-50, 100])
geometry = sections.MergedSection([rec1, rec2])
mesh = geometry.create_mesh(mesh_sizes=[5, 2.5])
section = CrossSection(geometry, mesh)
section.display_mesh_info()

>>>Mesh Statistics:
```

(continues on next page)

(continued from previous page)

```
>>>--4920 nodes
>>>--2365 elements
>>>--2 regions
```

display_results(*fmt='8.6e'*)

Prints the results that have been calculated to the terminal.

Parameters **fmt** (*string*) – Number formatting string

The following example displays the geometric section properties for a 100D x 50W rectangle with three digits after the decimal point:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.RectangularSection(d=100, b=50)
mesh = geometry.create_mesh(mesh_sizes=[5])

section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()

section.display_results(fmt='.3f')
```

get_As()

Returns Shear area for loading about the centroidal axis *(A_sx, A_sy)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
(A_sx, A_sy) = section.get_As()
```

get_As_p()

Returns Shear area for loading about the principal bending axis *(A_s11, A_s22)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
(A_s11, A_s22) = section.get_As_p()
```

get_area()

Returns Cross-section area

Return type float

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
area = section.get_area()
```

get_beta()

Returns Monosymmetry constant for bending about both global axes *(beta_x_plus, beta_x_minus, beta_y_plus, beta_y_minus)*. The *plus* value relates to the top flange in compression and the *minus* value relates to the bottom flange in compression.

Return type tuple(float, float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
```

(continues on next page)

(continued from previous page)

```
section.calculate_warping_properties()
(beta_x_plus, beta_x_minus, beta_y_plus, beta_y_minus) = section.get_
↪beta()
```

get_beta_p()

Returns Monosymmetry constant for bending about both principal axes *(beta_11_plus, beta_11_minus, beta_22_plus, beta_22_minus)*. The *plus* value relates to the top flange in compression and the *minus* value relates to the bottom flange in compression.

Return type tuple(float, float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
(beta_11_plus, beta_11_minus, beta_22_plus, beta_22_minus) = section.
↪get_beta_p()
```

get_c()

Returns Elastic centroid *(cx, cy)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(cx, cy) = section.get_c()
```

get_ea()

Returns Modulus weighted area (axial rigidity)

Return type float

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
ea = section.get_ea()
```

get_gamma()

Returns Warping constant

Return type float

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
gamma = section.get_gamma()
```

get_ic()

Returns Second moments of area centroidal axis *(ixx_c, iyy_c, ixy_c)*

Return type tuple(float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(ixx_c, iyy_c, ixy_c) = section.get_ic()
```

get_ig()

Returns Second moments of area about the global axis *(ixx_g, iyy_g, ixy_g)*

Return type tuple(float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(ixx_g, iyy_g, ixy_g) = section.get_ig()
```

get_ip()

Returns Second moments of area about the principal axis *(i11_c, i22_c)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(i11_c, i22_c) = section.get_ip()
```

get_j()

Returns St. Venant torsion constant

Return type float

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
j = section.get_j()
```

get_pc()

Returns Centroidal axis plastic centroid *(x_pc, y_pc)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
(x_pc, y_pc) = section.get_pc()
```

get_pc_p()

Returns Principal bending axis plastic centroid *(x11_pc, y22_pc)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
(x11_pc, y22_pc) = section.get_pc_p()
```

get_perimeter()

Returns Cross-section perimeter

Return type float

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
perimeter = section.get_perimeter()
```

get_phi()

Returns Principal bending axis angle

Return type float

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
phi = section.get_phi()
```

get_q()

Returns First moments of area about the global axis *(qx, qy)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(qx, qy) = section.get_q()
```

get_rc()

Returns Radii of gyration about the centroidal axis *(rx, ry)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(rx, ry) = section.get_rc()
```

get_rp()

Returns Radii of gyration about the principal axis *(r11, r22)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(r11, r22) = section.get_rp()
```

get_s()

Returns Plastic section moduli about the centroidal axis *(sxx, syy)*

Return type tuple(float, float)

If material properties have been specified, returns the plastic moment $M_p = f_y S$.

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
(sxx, syy) = section.get_s()
```

get_sc()

Returns Centroidal axis shear centre (elasticity approach) *(x_se, y_se)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
(x_se, y_se) = section.get_sc()
```

get_sc_p()

Returns Principal axis shear centre (elasticity approach) *(x11_se, y22_se)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
(x11_se, y22_se) = section.get_sc_p()
```

get_sc_t()

Returns Centroidal axis shear centre (Trefftz's approach) *(x_st, y_st)*

Return type tuple(float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
```

(continues on next page)

(continued from previous page)

```
section.calculate_warping_properties()
(x_st, y_st) = section.get_sc_t()
```

get_sf()

Returns Centroidal axis shape factors with respect to the top and bottom fibres *(sf_xx_plus, sf_xx_minus, sf_yy_plus, sf_yy_minus)*

Return type tuple(float, float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
(sf_xx_plus, sf_xx_minus, sf_yy_plus, sf_yy_minus) = section.get_sf()
```

get_sf_p()

Returns Principal bending axis shape factors with respect to the top and bottom fibres *(sf_11_plus, sf_11_minus, sf_22_plus, sf_22_minus)*

Return type tuple(float, float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
(sf_11_plus, sf_11_minus, sf_22_plus, sf_22_minus) = section.get_sf_
↪p()
```

get_sp()

Returns Plastic section moduli about the principal bending axis *(s11, s22)*

Return type tuple(float, float)

If material properties have been specified, returns the plastic moment $M_p = f_y S$.

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_plastic_properties()
(s11, s22) = section.get_sp()
```

get_z()

Returns Elastic section moduli about the centroidal axis with respect to the top and bottom fibres *(zxx_plus, zxx_minus, zyy_plus, zyy_minus)*

Return type tuple(float, float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(zxx_plus, zxx_minus, zyy_plus, zyy_minus) = section.get_z()
```

get_zp()

Returns Elastic section moduli about the principal axis with respect to the top and bottom fibres *(z11_plus, z11_minus, z22_plus, z22_minus)*

Return type tuple(float, float, float, float)

```
section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
(z11_plus, z11_minus, z22_plus, z22_minus) = section.get_zp()
```

plot_centroids(*pause=True*)

Plots the elastic centroid, the shear centre, the plastic centroids and the principal axis, if they have been calculated, on top of the finite element mesh.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example analyses a 200 PFC section and displays a plot of the centroids:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.PfcSection(d=200, b=75, t_f=12, t_w=6, r=12, n_
↪r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])

section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
section.calculate_plastic_properties()

section.plot_centroids()
```

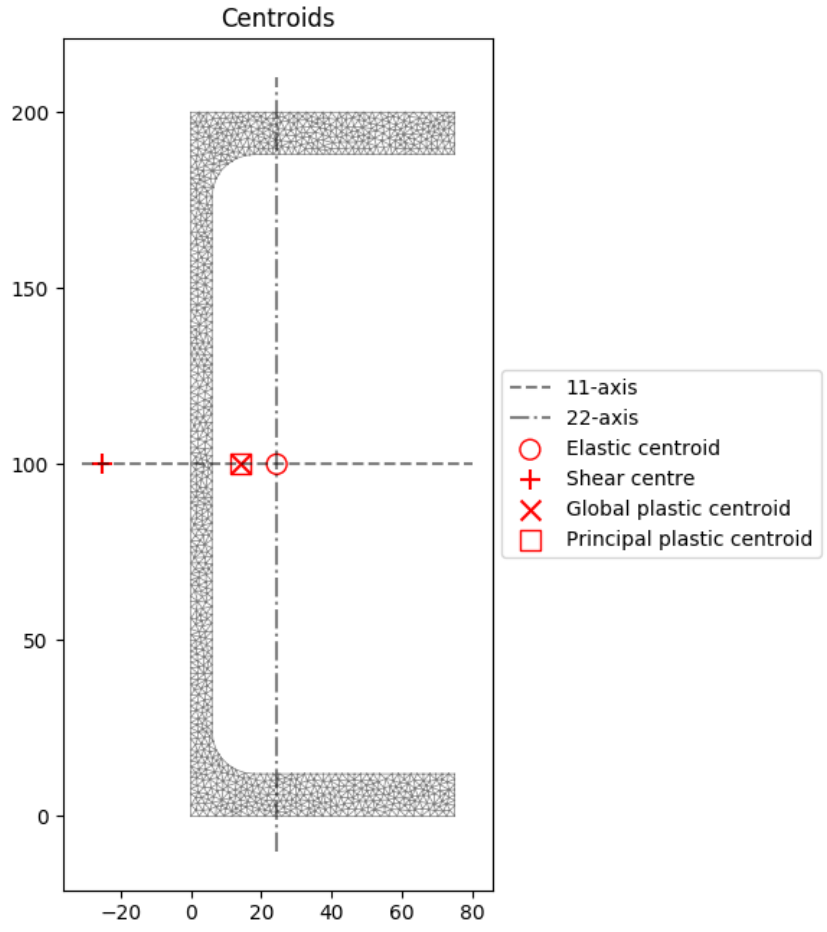


Fig. 87: Plot of the centroids generated by the above example.

The following example analyses a 150x90x12 UA section and displays a plot of the centroids:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_
↪r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])

section = CrossSection(geometry, mesh)
section.calculate_geometric_properties()
section.calculate_warping_properties()
section.calculate_plastic_properties()

section.plot_centroids()
```

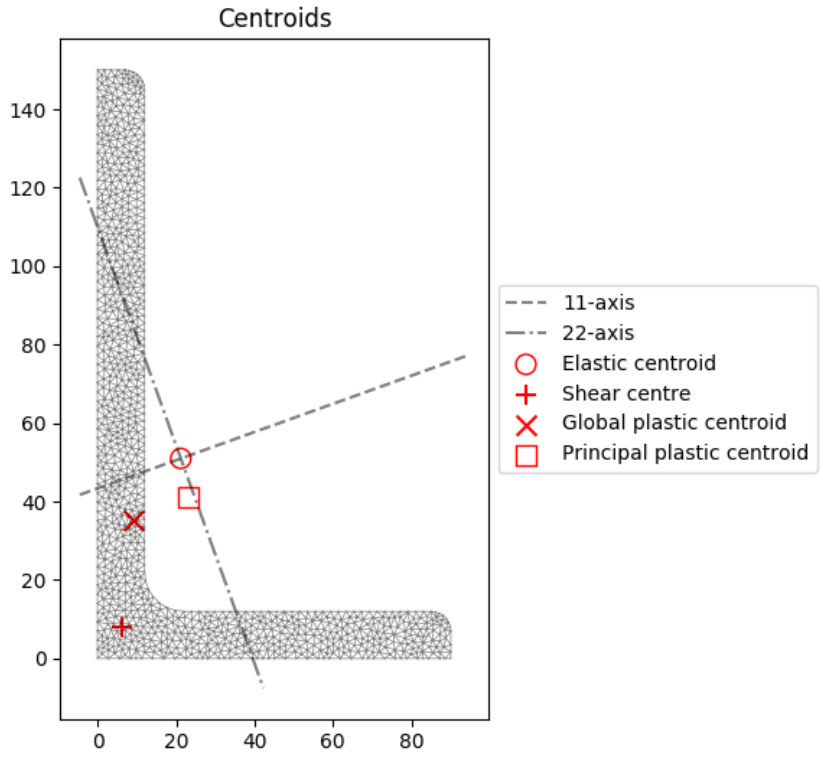


Fig. 88: Plot of the centroids generated by the above example.

plot_mesh(*ax=None*, *pause=True*, *alpha=1*, *materials=False*, *mask=None*)

Plots the finite element mesh. If no axes object is supplied a new figure and axis is created.

Parameters

- **ax** (`matplotlib.axes.Axes`) – Axes object on which the mesh is plotted
- **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.
- **alpha** (*float*) – Transparency of the mesh outlines: $0 \leq \alpha \leq 1$
- **materials** (*bool*) – If set to true and material properties have been provided to the *CrossSection* object, shades the elements with the specified material colours
- **mask** (*list[bool]*) – Mask array, of length `num_nodes`, to mask out triangles

Returns Matplotlib figure and axes objects (fig, ax)

Return type (`matplotlib.figure.Figure`, `matplotlib.axes`)

The following example plots the mesh generated for the second example listed under the *CrossSection* object definition:

```
import sectionproperties.pre.sections as sections
from sectionproperties.pre.pre import Material
from sectionproperties.analysis.cross_section import CrossSection

geom_steel = sections.RectangularSection(d=50, b=50)
geom_timber = sections.RectangularSection(d=50, b=50, shift=[50, 0])
geometry = sections.MergedSection([geom_steel, geom_timber])
geometry.clean_geometry()

mesh = geometry.create_mesh(mesh_sizes=[5, 10])

steel = Material(
    name='Steel', elastic_modulus=200e3, poissons_ratio=0.3, yield_
↪strength=250,
    color='grey'
)
timber = Material(
    name='Timber', elastic_modulus=8e3, poissons_ratio=0.35, yield_
↪strength=20,
    color='burlywood'
)

section = CrossSection(geometry, mesh, [steel, timber])
section.plot_mesh(materials=True, alpha=0.5)
```

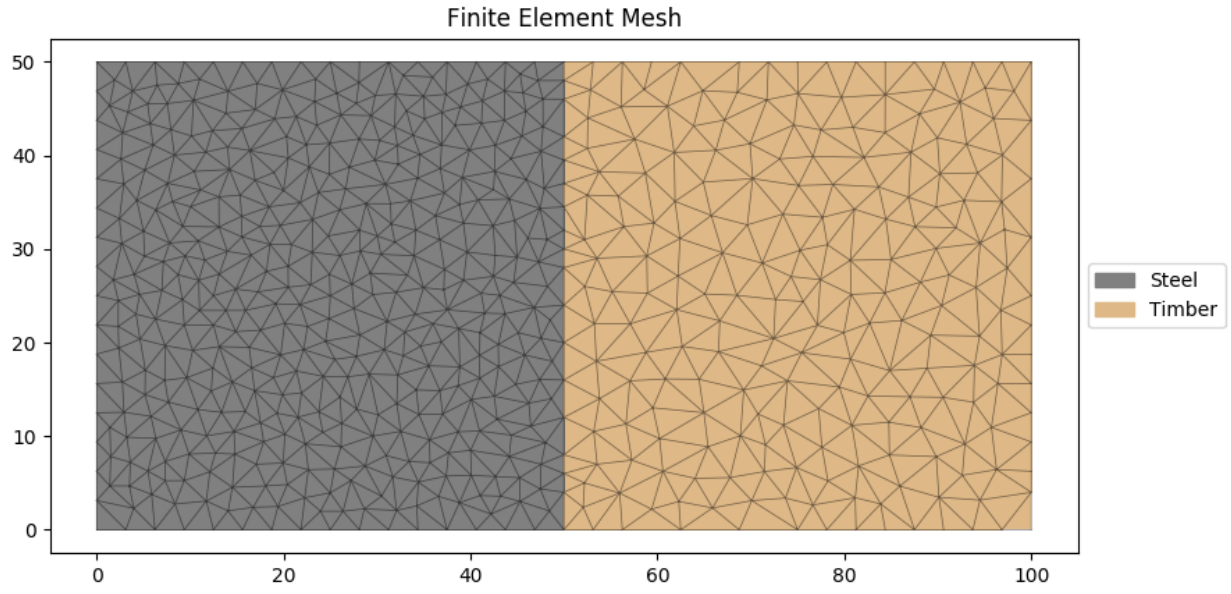


Fig. 89: Finite element mesh generated by the above example.

PlasticSection Class

class sectionproperties.analysis.cross_section.**PlasticSection**(*geometry*, *materials*, *debug*)

Bases: object

Class for the plastic analysis of cross-sections.

Stores the finite element geometry and material information and provides methods to compute the plastic section properties.

Parameters

- **geometry** (*Geometry*) – Cross-section geometry object
- **materials** (list[*Material*]) – A list of material properties corresponding to various regions in the geometry and mesh.
- **debug** (*bool*) – If set to True, the geometry is plotted each time a new mesh is generated by the plastic centroid algorithm.

Variables

- **geometry** (*Geometry*) – Deep copy of the cross-section geometry object provided to the constructor
- **materials** (list[*Material*]) – A list of material properties corresponding to various regions in the geometry and mesh.
- **debug** (*bool*) – If set to True, the geometry is plotted each time a new mesh is generated by the plastic centroid algorithm.
- **mesh** (meshpy.triangle.MeshInfo) – Mesh object returned by meshpy
- **mesh_nodes** (numpy.ndarray) – Array of node coordinates from the mesh
- **mesh_elements** (numpy.ndarray) – Array of connectivities from the mesh
- **elements** (list[*Tri6*]) – List of finite element objects describing the cross-section mesh
- **f_top** (*float*) – Current force in the top region
- **c_top** – Centroid of the force in the top region *(c_top_x, c_top_y)*
- **c_bot** – Centroid of the force in the bottom region *(c_bot_x, c_bot_y)*

add_line(*geometry*, *line*)

Adds a line a geometry object. Finds the intersection points of the line with the current facets and splits the existing facets to accomodate the new line.

Parameters

- **geometry** (*Geometry*) – Cross-section geometry object used to generate the mesh
- **line** (list[numpy.ndarray, numpy.ndarray]) – A point p and a unit vector u defining a line to add to the mesh (line: p -> p + u)

calculate_centroid(*elements*)

Calculates the elastic centroid from a list of finite elements.

Parameters **elements** (list[*Tri6*]) – A list of Tri6 finite elements.

Returns A tuple containing the x and y location of the elastic centroid.

Return type tuple(float, float)

calculate_extreme_fibres(*angle*)

Calculates the locations of the extreme fibres along and perpendicular to the axis specified by 'angle' using the elements stored in *self.elements*.

Parameters **angle** (*float*) – Angle (in radians) along which to calculate the extreme fibre locations

Returns The location of the extreme fibres parallel (u) and perpendicular (v) to the axis *(u_min, u_max, v_min, v_max)*

Return type tuple(float, float, float, float)

calculate_plastic_force(*elements*, *u*, *p*)

Sums the forces above and below the axis defined by unit vector *u* and point *p*. Also returns the force centroid of the forces above and below the axis.

Parameters

- **elements** (list[*Tri6*]) – A list of Tri6 finite elements.
- **u** (numpy.ndarray) – Unit vector defining the direction of the axis
- **p** (numpy.ndarray) – Point on the axis

Returns Force in the top and bottom areas *(f_top, f_bot)*

Return type tuple(float, float)

calculate_plastic_properties(*cross_section*, *verbose*)

Calculates the location of the plastic centroid with respect to the centroidal and principal bending axes, the plastic section moduli and shape factors and stores the results to the supplied *CrossSection* object.

Parameters

- **cross_section** (*CrossSection*) – Cross section object that uses the same geometry and materials specified in the class constructor
- **verbose** (*bool*) – If set to True, the number of iterations required for each plastic axis is printed to the terminal.

check_convergence(*root_result*, *axis*)

Checks that the function solver converged and if not, raises a helpful error.

Parameters

- **root_result** (scipy.optimize.RootResults) – Result object from the root finder
- **axis** (*string*) – Axis being considered by the function sovler

Raises **RuntimeError** – If the function solver did not converge

create_plastic_mesh(*new_line=None*)

Generates a triangular mesh of a deep copy of the geometry stored in *self.geometry*. Optionally, a line can be added to the copied geometry, which is defined by a point *p* and a unit vector *u*.

Parameters

- **new_line** (list[numpy.ndarray, numpy.ndarray]) – A point p and a unit vector u defining a line to add to the mesh (new_line: p -> p + u) [*p*, *u*]
- **mesh** (meshpy.triangle.MeshInfo) – Mesh object returned by meshpy

evaluate_force_eq(*d*, *u*, *u_p*, *verbose*)

Given a distance *d* from the centroid to an axis (defined by unit vector *u*), creates a mesh including the new and axis and calculates the force equilibrium. The resultant force, as a ratio of the total force, is returned.

Parameters

- **d** (*float*) – Distance from the centroid to current axis
- **u** (numpy.ndarray) – Unit vector defining the direction of the axis
- **u_p** (numpy.ndarray) – Unit vector perpendicular to the direction of the axis
- **verbose** (*bool*) – If set to True, the number of iterations required for each plastic axis is printed to the terminal.

Returns The force equilibrium norm

Return type float

get_elements(*mesh*)

Extracts finite elements from the provided mesh and returns Tri6 finite elements with their associated material properties.

Parameters **mesh** (meshpy.triangle.MeshInfo) – Mesh object returned by meshpy

Returns A tuple containing an array of the nodes locations, element indicies and a list of the finite elements.

Return type tuple(numpy.ndarray, numpy.ndarray, list[*Tri6*])

pc_algorithm(*u*, *dlim*, *axis*, *verbose*)

An algorithm used for solving for the location of the plastic centroid. The algorithm searches for the location of the axis, defined by unit vector *u* and within the section depth, that satisfies force equilibrium.

Parameters

- **u** (numpy.ndarray) – Unit vector defining the direction of the axis
- **dlim** (*list[float, float]*) – List [dmax, dmin] containing the distances from the centroid to the extreme fibres perpendicular to the axis
- **axis** (*int*) – The current axis direction: 1 (e.g. x or 11) or 2 (e.g. y or 22)
- **verbose** (*bool*) – If set to True, the number of iterations required for each plastic axis is printed to the terminal.

Returns The distance to the plastic centroid axis *d*, the result object *r*, the force in the top of the section *f_top* and the location of the centroids of the top and bottom areas *c_top* and *c_bottom*

Return type tuple(float, scipy.optimize.RootResults, float, list[float, float], list[float, float])

plot_mesh(*nodes*, *elements*, *element_list*, *materials*)

Watered down implementation of the CrossSection method to plot the finite element mesh, showing material properties.

point_within_element(*pt*)

Determines whether a point lies within an element in the mesh stored in *self.mesh_elements*.

Parameters **pt** (numpy.ndarray) – Point to check

Returns Whether the point lies within an element

Return type bool

print_verbose(*d*, *root_result*, *axis*)

Prints information related to the function solver convergence to the terminal.

Parameters

- **d** (*float*) – Location of the plastic centroid axis
- **root_result** (scipy.optimize.RootResults) – Result object from the root finder
- **axis** (*string*) – Axis being considered by the function sovler

rebuild_parent_facet(*geometry*, *fct_idx*, *pt_idx*)

Splits and rebuilds a facet at a given point.

Parameters

- **geometry** (*Geometry*) – Cross-section geometry object used to generate the mesh
- **fct_idx** (*int*) – Index of the facet to be split
- **pt_idx** (*int*) – Index of the point to insert into the facet

StressPost Class

class sectionproperties.analysis.cross_section.**StressPost**(*cross_section*)
Bases: object

Class for post-processing finite element stress results.

A StressPost object is created when a stress analysis is carried out and is returned as an object to allow post-processing of the results. The StressPost object creates a deep copy of the MaterialGroups within the cross-section to allow the calculation of stresses for each material. Methods for post-processing the calculated stresses are provided.

Parameters **cross_section** (*CrossSection*) – Cross section object for stress calculation

Variables

- **cross_section** (*CrossSection*) – Cross section object for stress calculation
- **material_groups** (list[MaterialGroup]) – A deep copy of the *cross_section* material groups to allow a new stress analysis

get_stress()
Returns the stresses within each material belonging to the current *StressPost* object.

Returns A list of dictionaries containing the cross-section stresses for each material.

Return type list[dict]

A dictionary is returned for each material in the cross-section, containing the following keys and values:

- *'Material'*: Material name
- *'sig_zz_n'*: Normal stress $\sigma_{zz,N}$ resulting from the axial load N
- *'sig_zz_mxx'*: Normal stress $\sigma_{zz,Mxx}$ resulting from the bending moment M_{xx}
- *'sig_zz_myy'*: Normal stress $\sigma_{zz,Myy}$ resulting from the bending moment M_{yy}
- *'sig_zz_m11'*: Normal stress $\sigma_{zz,M11}$ resulting from the bending moment M_{11}
- *'sig_zz_m22'*: Normal stress $\sigma_{zz,M22}$ resulting from the bending moment M_{22}
- *'sig_zz_m'*: Normal stress $\sigma_{zz,\Sigma M}$ resulting from all bending moments
- *'sig_zx_mzz'*: *x*-component of the shear stress $\sigma_{zx,Mzz}$ resulting from the torsion moment
- *'sig_zy_mzz'*: *y*-component of the shear stress $\sigma_{zy,Mzz}$ resulting from the torsion moment
- *'sig_zxy_mzz'*: Resultant shear stress $\sigma_{zxy,Mzz}$ resulting from the torsion moment
- *'sig_zx_vx'*: *x*-component of the shear stress $\sigma_{zx,Vx}$ resulting from the shear force V_x
- *'sig_zy_vx'*: *y*-component of the shear stress $\sigma_{zy,Vx}$ resulting from the shear force V_x
- *'sig_zxy_vx'*: Resultant shear stress $\sigma_{zxy,Vx}$ resulting from the shear force V_x
- *'sig_zx_vy'*: *x*-component of the shear stress $\sigma_{zx,Vy}$ resulting from the shear force V_y
- *'sig_zy_vy'*: *y*-component of the shear stress $\sigma_{zy,Vy}$ resulting from the shear force V_y
- *'sig_zxy_vy'*: Resultant shear stress $\sigma_{zxy,Vy}$ resulting from the shear force V_y
- *'sig_zx_v'*: *x*-component of the shear stress $\sigma_{zx,\Sigma V}$ resulting from all shear forces
- *'sig_zy_v'*: *y*-component of the shear stress $\sigma_{zy,\Sigma V}$ resulting from all shear forces
- *'sig_zxy_v'*: Resultant shear stress $\sigma_{zxy,\Sigma V}$ resulting from all shear forces
- *'sig_zz'*: Combined normal stress σ_{zz} resulting from all actions

- *'sig_zx'*: x-component of the shear stress σ_{zx} resulting from all actions
- *'sig_zy'*: y-component of the shear stress σ_{zy} resulting from all actions
- *'sig_zxy'*: Resultant shear stress σ_{zxy} resulting from all actions
- *'sig_vm'*: von Mises stress σ_{vM} resulting from all actions

The following example returns the normal stress within a 150x90x12 UA section resulting from an axial force of 10 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(N=10e3)

stresses = stress_post.get_stress()
print('Material: {0}'.format(stresses[0]['Material']))
print('Axial Stresses: {0}'.format(stresses[0]['sig_zz_n']))

$ Material: default
$ Axial Stresses: [3.6402569 3.6402569 3.6402569 ... 3.6402569 3.6402569 3.
↪6402569]
```

plot_stress_contour (*sigs*, *title*, *pause*)

Plots filled stress contours over the finite element mesh.

Parameters

- **sigs** (list[numpy.ndarray]) – List of nodal stress values for each material
- **title** (*string*) – Plot title
- **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

plot_stress_m11_zz (*pause=True*)

Produces a contour plot of the normal stress $\sigma_{zz,M11}$ resulting from the bending moment M_{11}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the normal stress within a 150x90x12 UA section resulting from a bending moment about the 11-axis of 5 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
```

(continues on next page)

(continued from previous page)

```
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(M11=5e6)

stress_post.plot_stress_m11_zz()
```

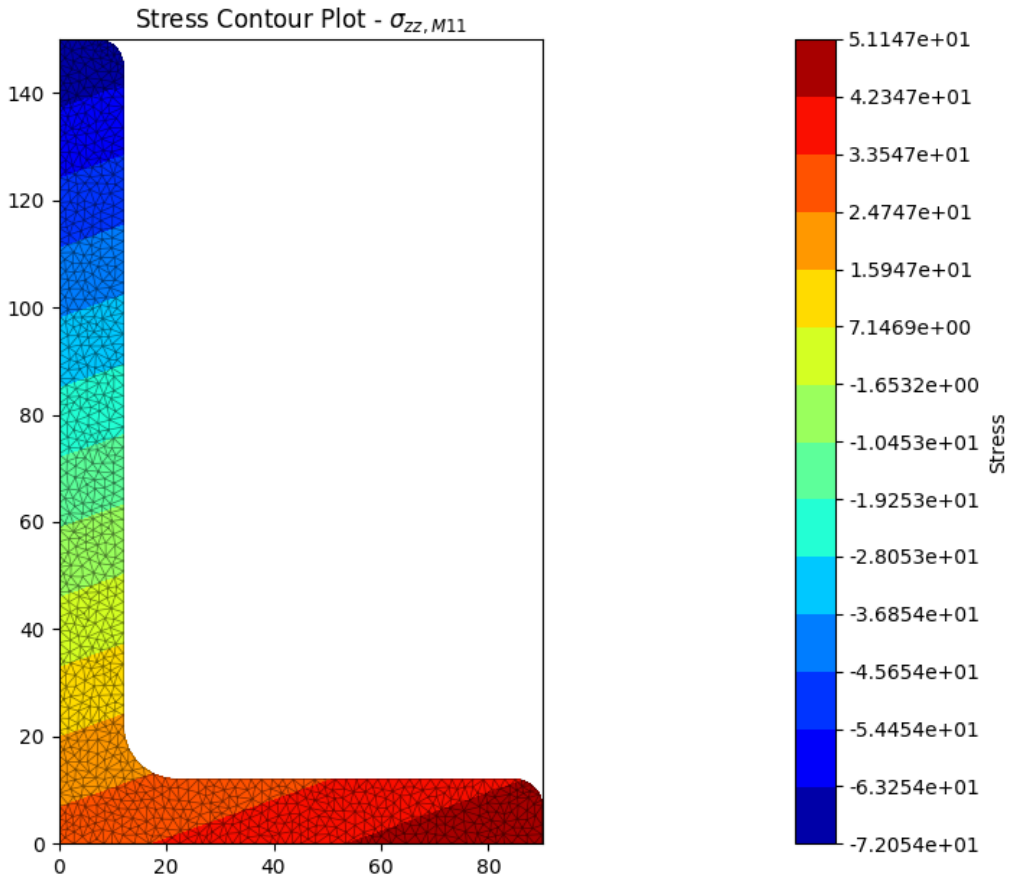


Fig. 90: Contour plot of the bending stress.

plot_stress_m22_zz (*pause=True*)

Produces a contour plot of the normal stress $\sigma_{zz,M22}$ resulting from the bending moment M_{22}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the normal stress within a 150x90x12 UA section resulting from a bending moment about the 22-axis of 2 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(M22=5e6)

stress_post.plot_stress_m22_zz()
```

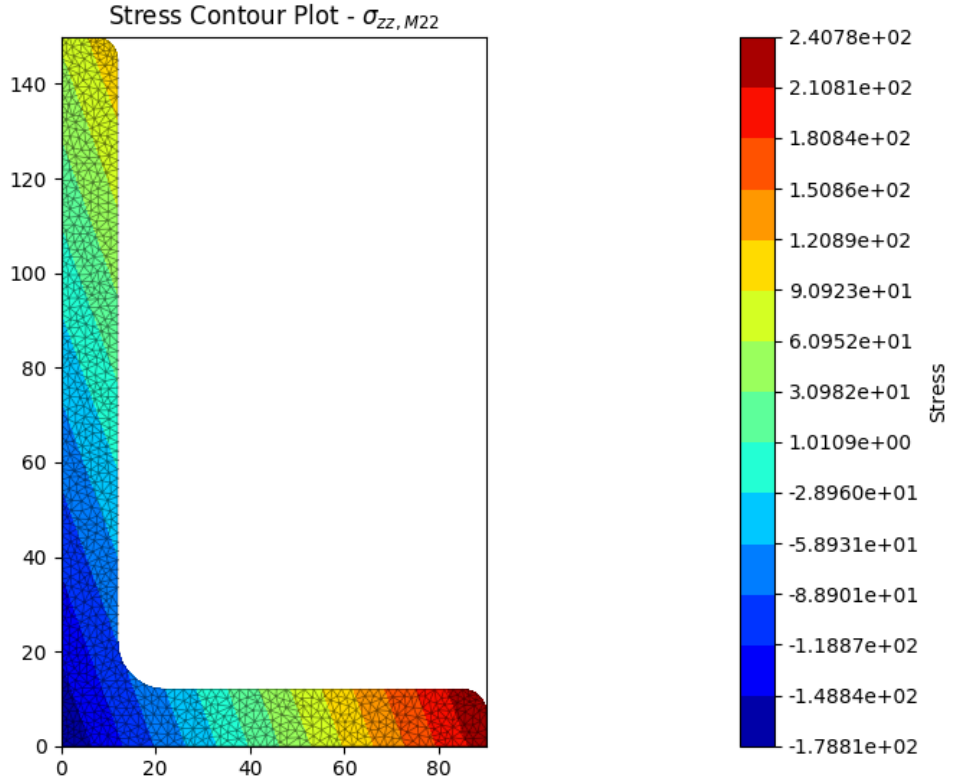


Fig. 91: Contour plot of the bending stress.

plot_stress_m_zz(*pause=True*)

Produces a contour plot of the normal stress $\sigma_{zz,\Sigma M}$ resulting from all bending moments $M_{xx} + M_{yy} + M_{11} + M_{22}$.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the normal stress within a 150x90x12 UA section resulting from a bending moment about the x-axis of 5 kN.m, a bending moment about the y-axis of 2 kN.m and a bending moment of 3 kN.m about the 11-axis:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mxx=5e6, Myy=2e6, M11=3e6)

stress_post.plot_stress_m_zz()
```

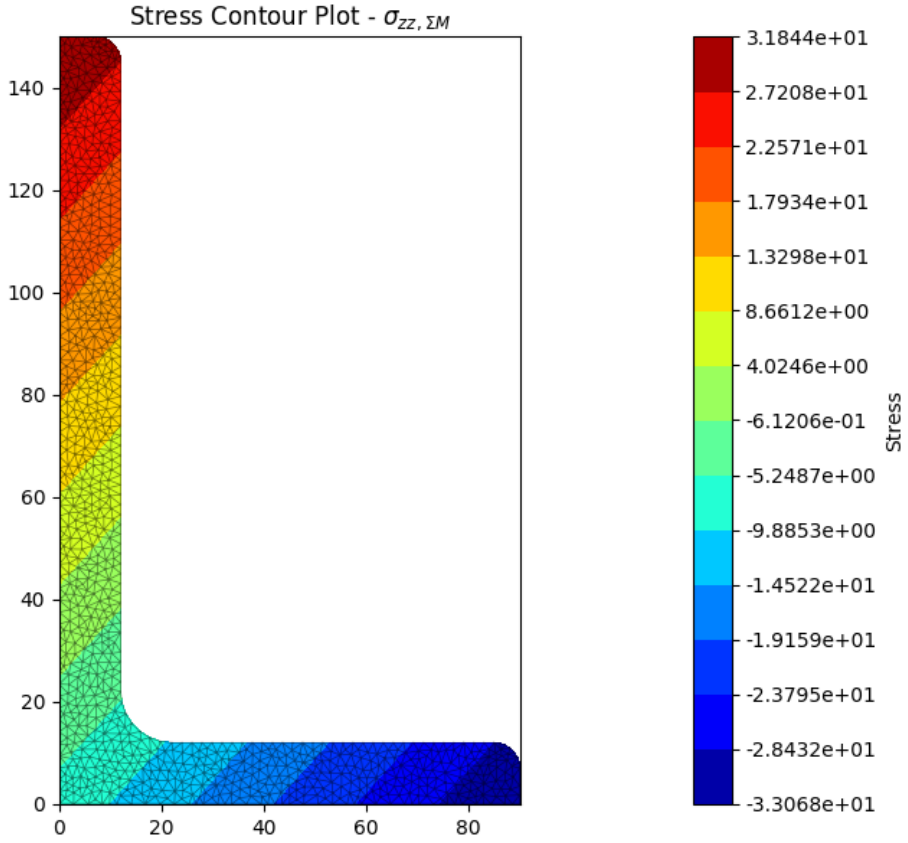


Fig. 92: Contour plot of the bending stress.

plot_stress_mxx_zz (*pause=True*)

Produces a contour plot of the normal stress $\sigma_{zz,Mxx}$ resulting from the bending moment M_{xx}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the normal stress within a 150x90x12 UA section resulting from a bending moment about the x-axis of 5 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mxx=5e6)

stress_post.plot_stress_mxx_zz()
```

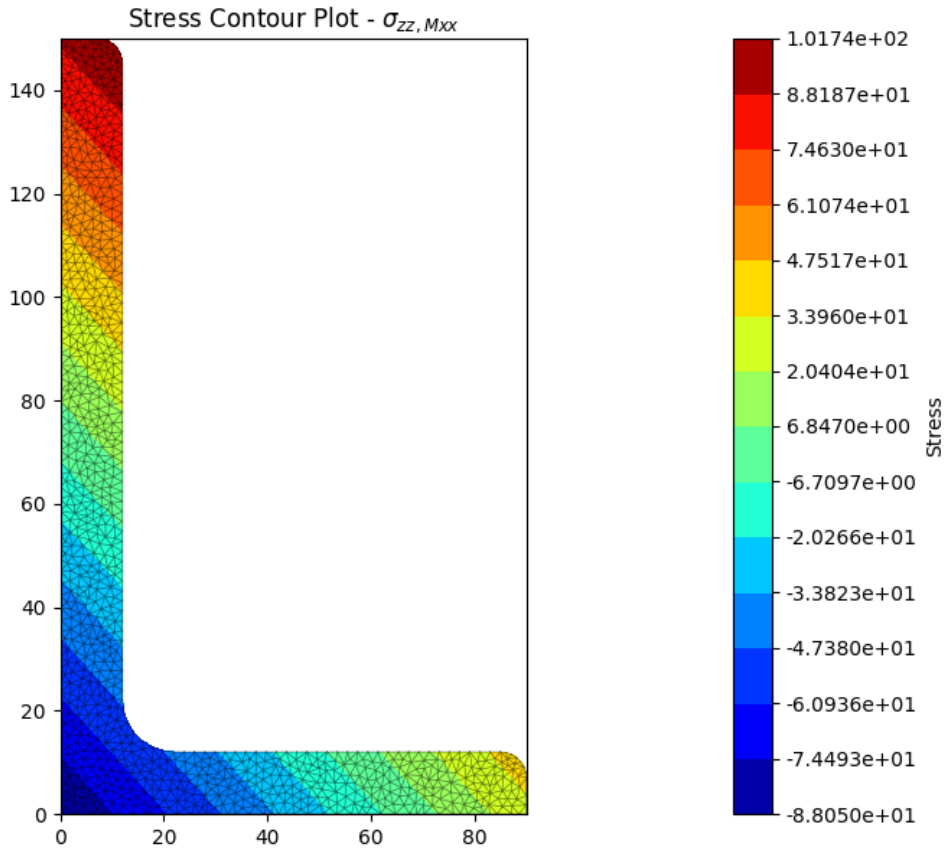


Fig. 93: Contour plot of the bending stress.

plot_stress_myy_zz (*pause=True*)

Produces a contour plot of the normal stress $\sigma_{zz,Myy}$ resulting from the bending moment M_{yy}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the normal stress within a 150x90x12 UA section resulting from a bending moment about the y-axis of 2 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Myy=2e6)

stress_post.plot_stress_myy_zz()
```

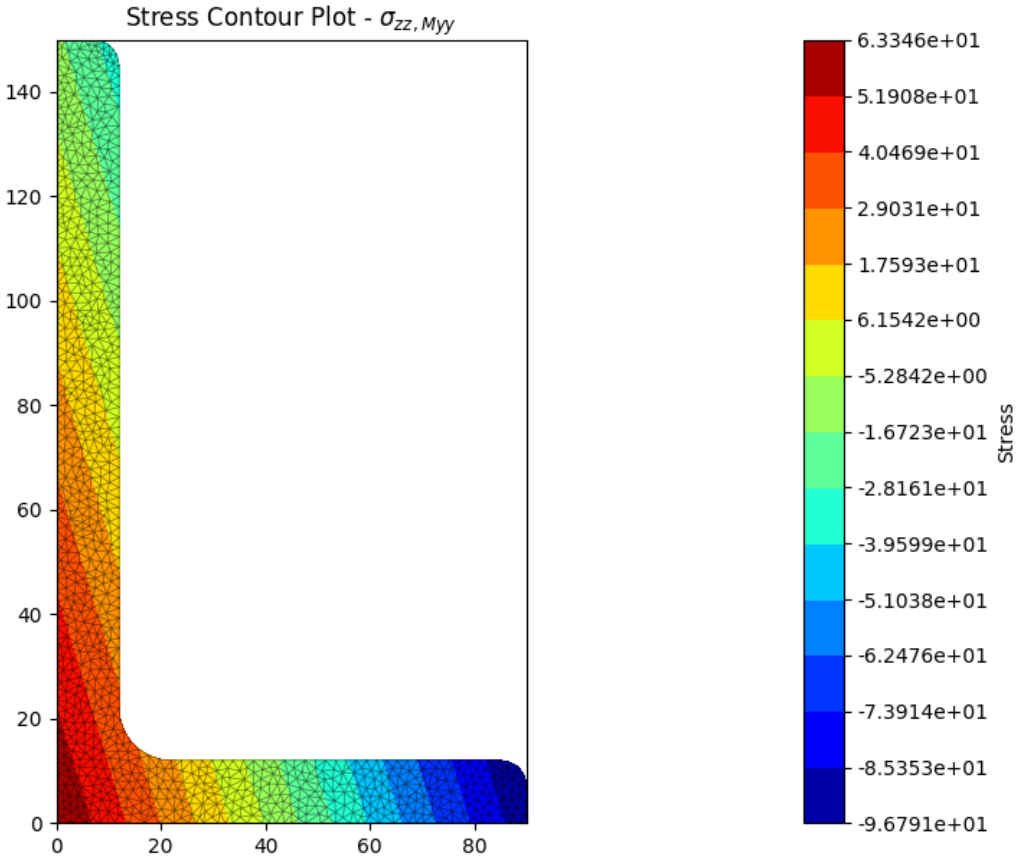


Fig. 94: Contour plot of the bending stress.

plot_stress_mzz_zx(*pause=True*)

Produces a contour plot of the x-component of the shear stress $\sigma_{zx,Mzz}$ resulting from the torsion moment M_{zz}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the x-component of the shear stress within a 150x90x12 UA section resulting

from a torsion moment of 1 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6)

stress_post.plot_stress_mzz_zx()
```

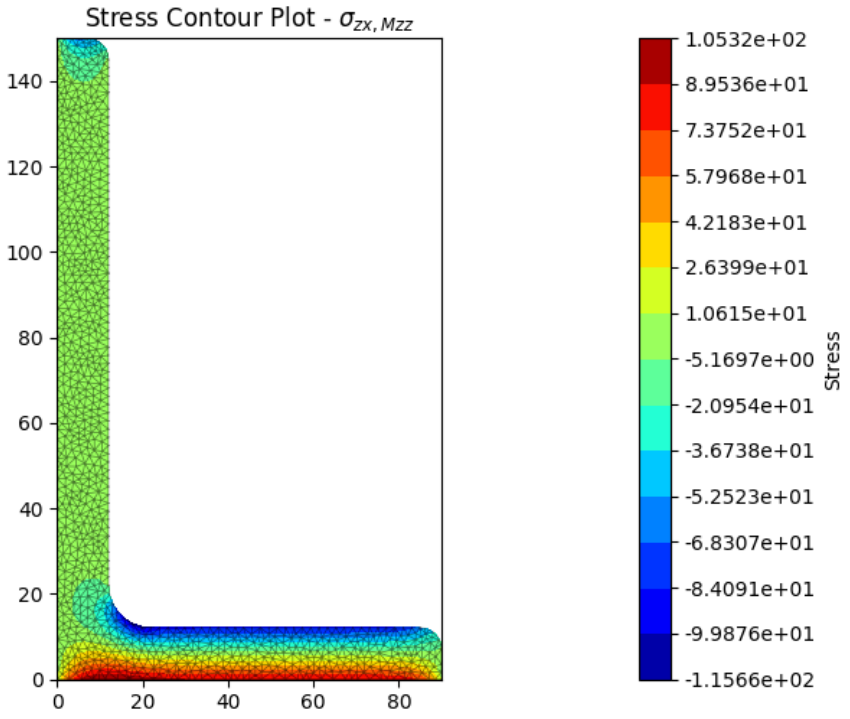


Fig. 95: Contour plot of the shear stress.

plot_stress_mzz_zxy (*pause=True*)

Produces a contour plot of the resultant shear stress $\sigma_{zxy,Mzz}$ resulting from the torsion moment M_{zz}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots a contour of the resultant shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection
```

(continues on next page)

(continued from previous page)

```
geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6)

stress_post.plot_stress_mzz_zxy()
```

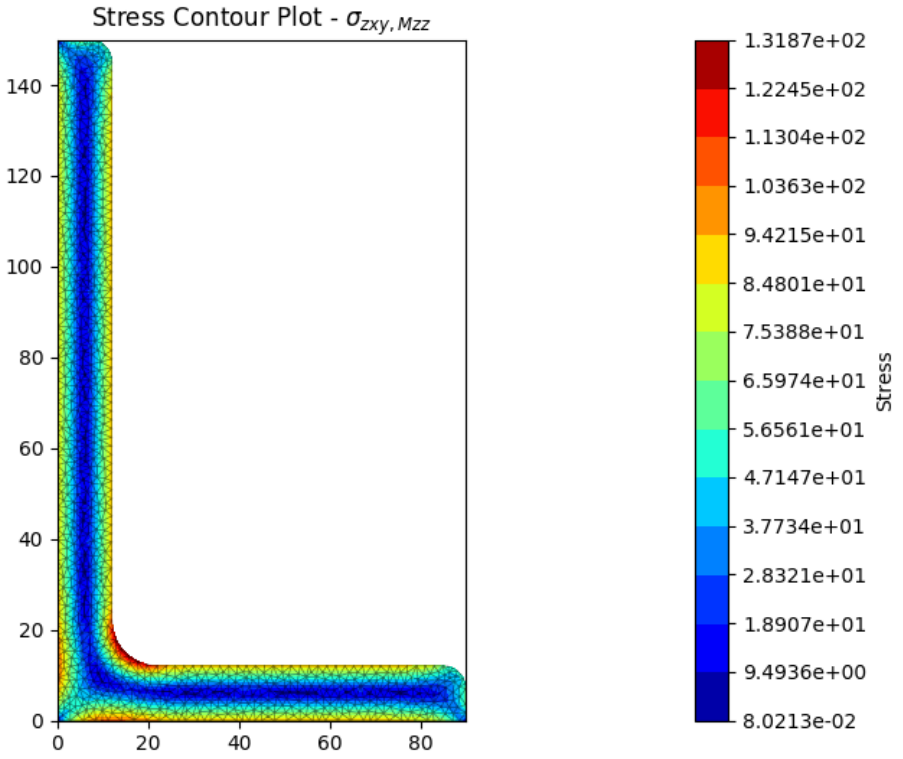


Fig. 96: Contour plot of the shear stress.

plot_stress_mzz_zy(*pause=True*)

Produces a contour plot of the y-component of the shear stress $\sigma_{zy,Mzz}$ resulting from the torsion moment M_{zz}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the y-component of the shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
```

(continues on next page)

(continued from previous page)

```
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6)

stress_post.plot_stress_mzz_zy()
```

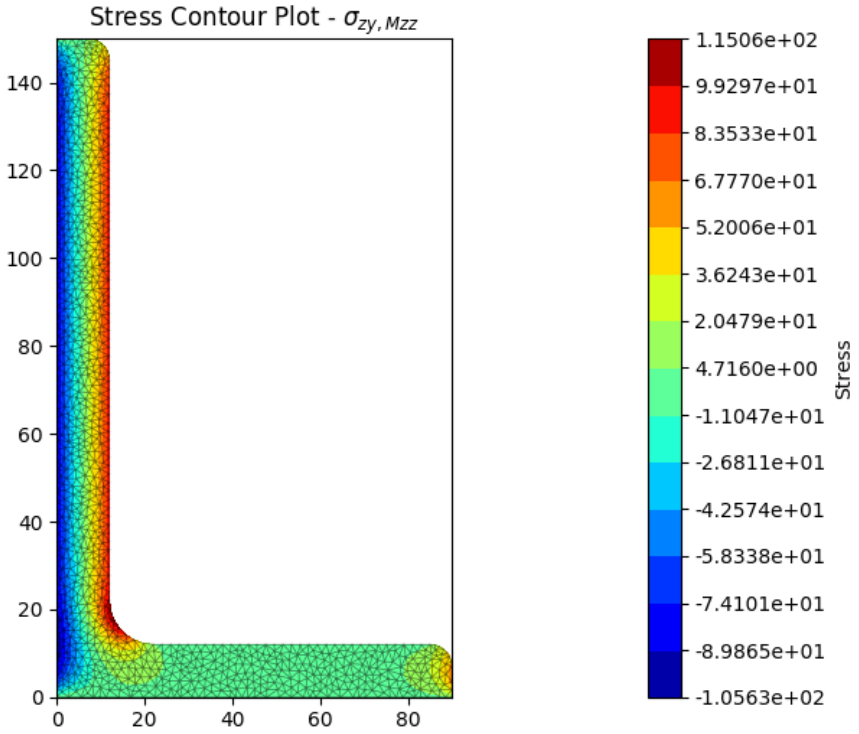


Fig. 97: Contour plot of the shear stress.

plot_stress_n_zz (*pause=True*)

Produces a contour plot of the normal stress $\sigma_{zz,N}$ resulting from the axial load N.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the normal stress within a 150x90x12 UA section resulting from an axial force of 10 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)
```

(continues on next page)

(continued from previous page)

```
section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(N=10e3)

stress_post.plot_stress_n_zz()
```

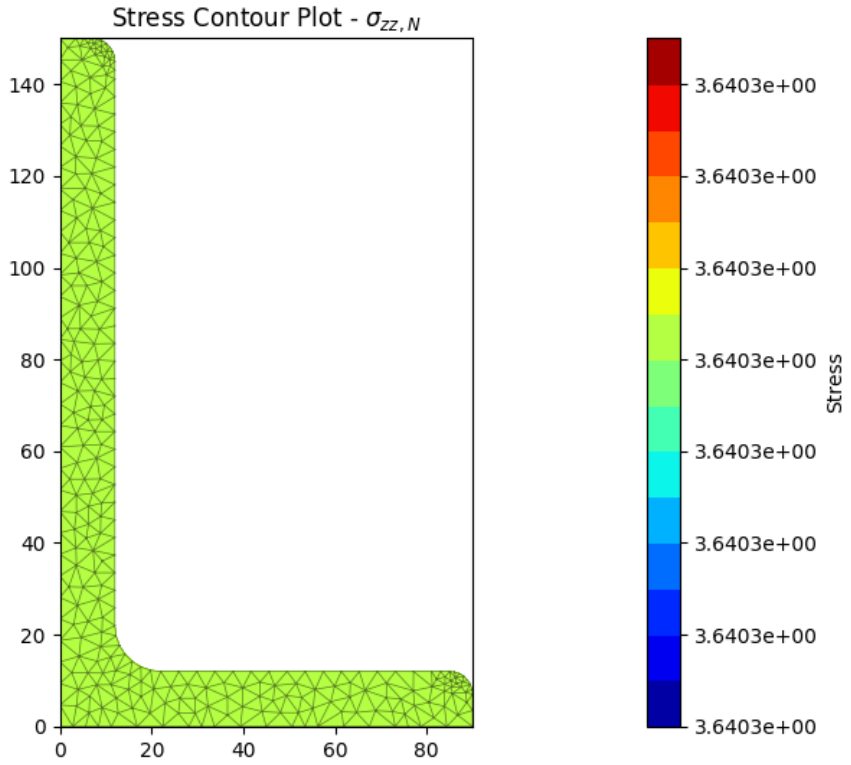


Fig. 98: Contour plot of the axial stress.

plot_stress_v_zx(*pause=True*)

Produces a contour plot of the x-component of the shear stress $\sigma_{zx,\Sigma V}$ resulting from the sum of the applied shear forces $V_x + V_y$.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the x-component of the shear stress within a 150x90x12 UA section resulting from a shear force of 15 kN in the x-direction and 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
```

(continues on next page)

(continued from previous page)

```
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3, Vy=30e3)

stress_post.plot_stress_v_zx()
```

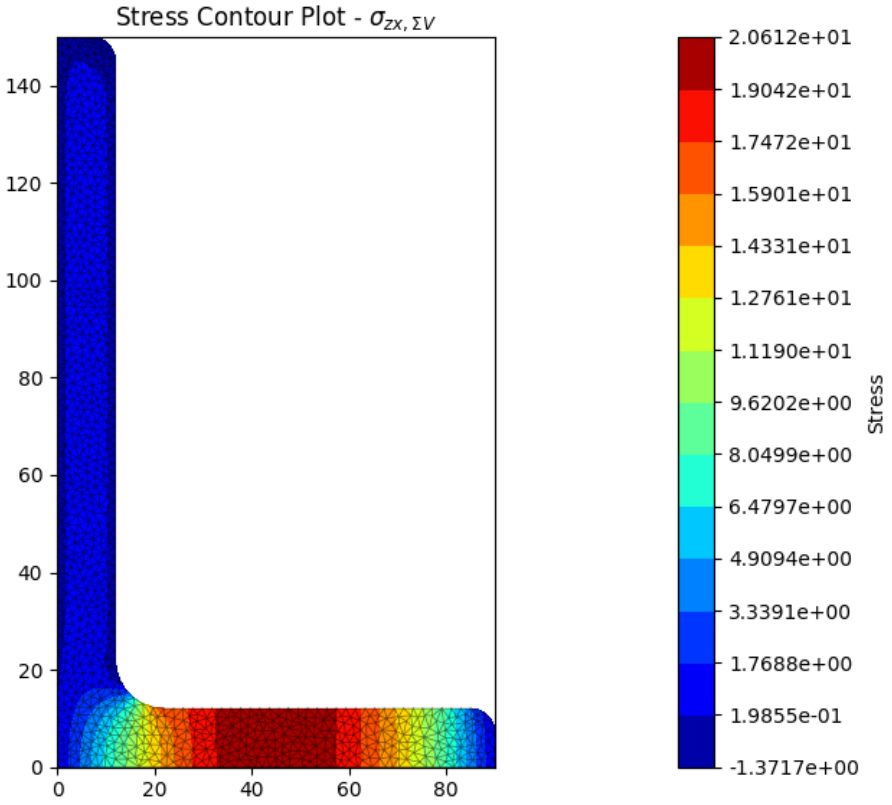


Fig. 99: Contour plot of the shear stress.

plot_stress_v_zxy(*pause=True*)

Produces a contour plot of the resultant shear stress $\sigma_{zxy,\Sigma V}$ resulting from the sum of the applied shear forces $V_x + V_y$.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots a contour of the resultant shear stress within a 150x90x12 UA section resulting from a shear force of 15 kN in the x-direction and 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

```

(continues on next page)

(continued from previous page)

```
section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3, Vy=30e3)

stress_post.plot_stress_v_zxy()
```

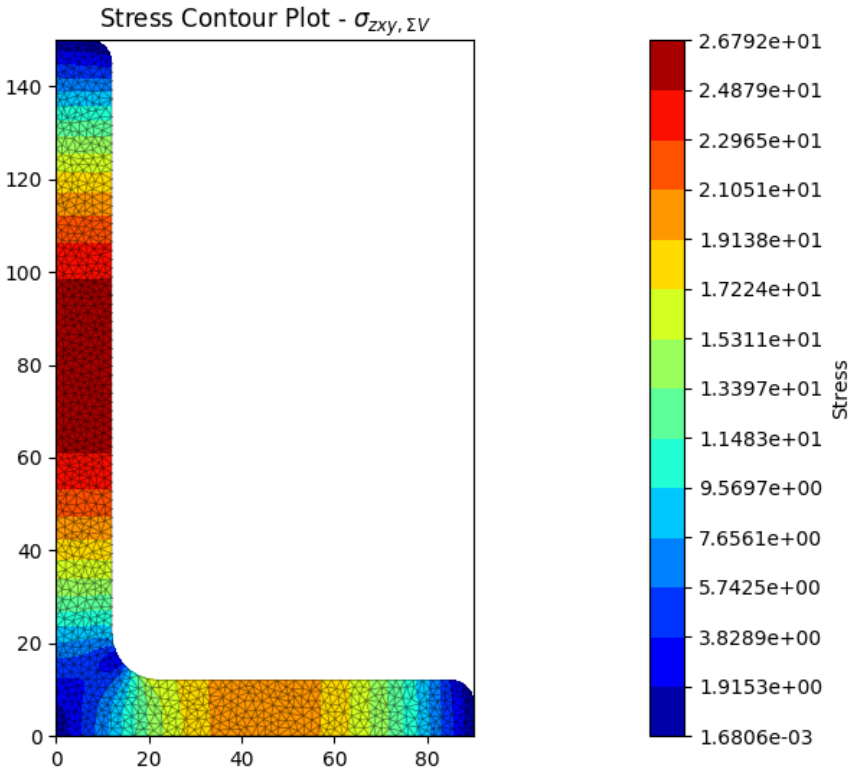


Fig. 100: Contour plot of the shear stress.

plot_stress_v_zy(*pause=True*)

Produces a contour plot of the y-component of the shear stress $\sigma_{zy,\Sigma V}$ resulting from the sum of the applied shear forces $V_x + V_y$.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the y-component of the shear stress within a 150x90x12 UA section resulting from a shear force of 15 kN in the x-direction and 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)
```

(continues on next page)

(continued from previous page)

```
section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3, Vy=30e3)

stress_post.plot_stress_v_zy()
```

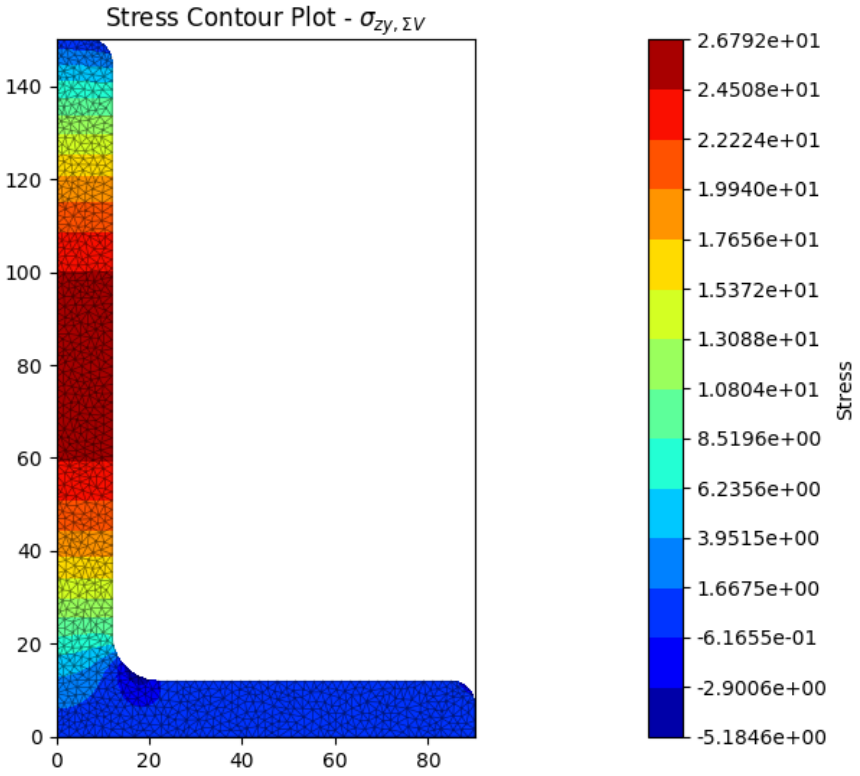


Fig. 101: Contour plot of the shear stress.

plot_stress_vector(*sigxs*, *sigys*, *title*, *pause*)
Plots stress vectors over the finite element mesh.

Parameters

- **sigxs** (list[numpy.ndarray]) – List of x-components of the nodal stress values for each material
- **sigys** (list[numpy.ndarray]) – List of y-components of the nodal stress values for each material
- **title** (*string*) – Plot title
- **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

plot_stress_vm(*pause=True*)
Produces a contour plot of the von Mises stress σ_{vM} resulting from all actions.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots a contour of the von Mises stress within a 150x90x12 UA section resulting from the following actions:

- $N = 50$ kN
- $M_{xx} = -5$ kN.m
- $M_{22} = 2.5$ kN.m
- $M_{zz} = 1.5$ kN.m
- $V_x = 10$ kN
- $V_y = 5$ kN

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(
    N=50e3, Mxx=-5e6, M22=2.5e6, Mzz=0.5e6, Vx=10e3, Vy=5e3
)

stress_post.plot_stress_vm()
```

plot_stress_vx_zx(*pause=True*)

Produces a contour plot of the x-component of the shear stress $\sigma_{zx,Vx}$ resulting from the shear force V_x.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the x-component of the shear stress within a 150x90x12 UA section resulting from a shear force in the x-direction of 15 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3)

stress_post.plot_stress_vx_zx()
```

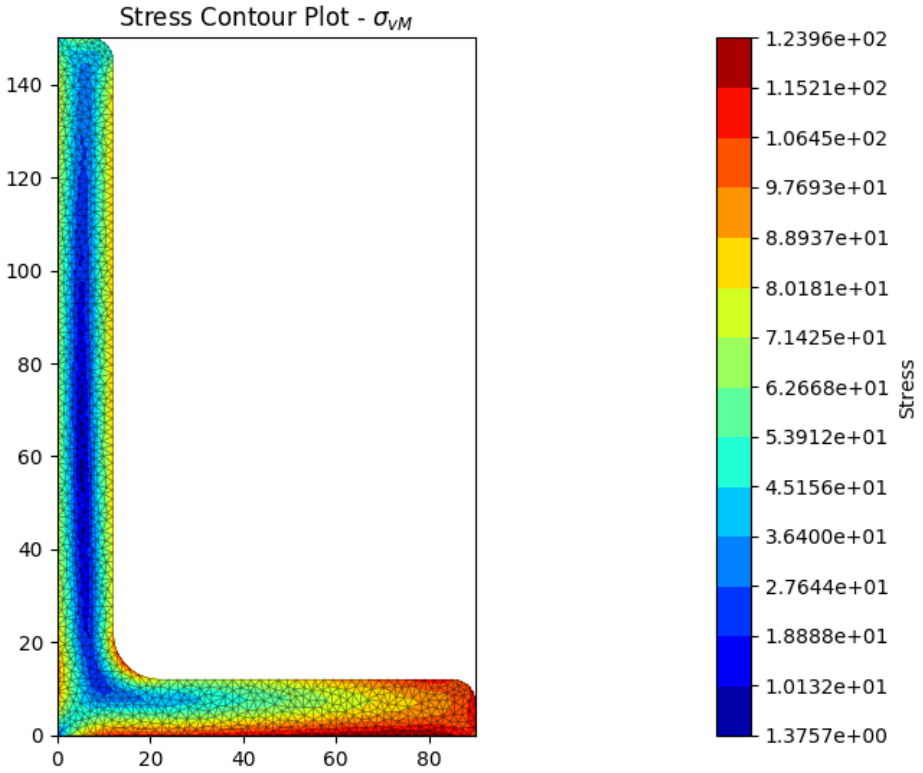


Fig. 102: Contour plot of the von Mises stress.

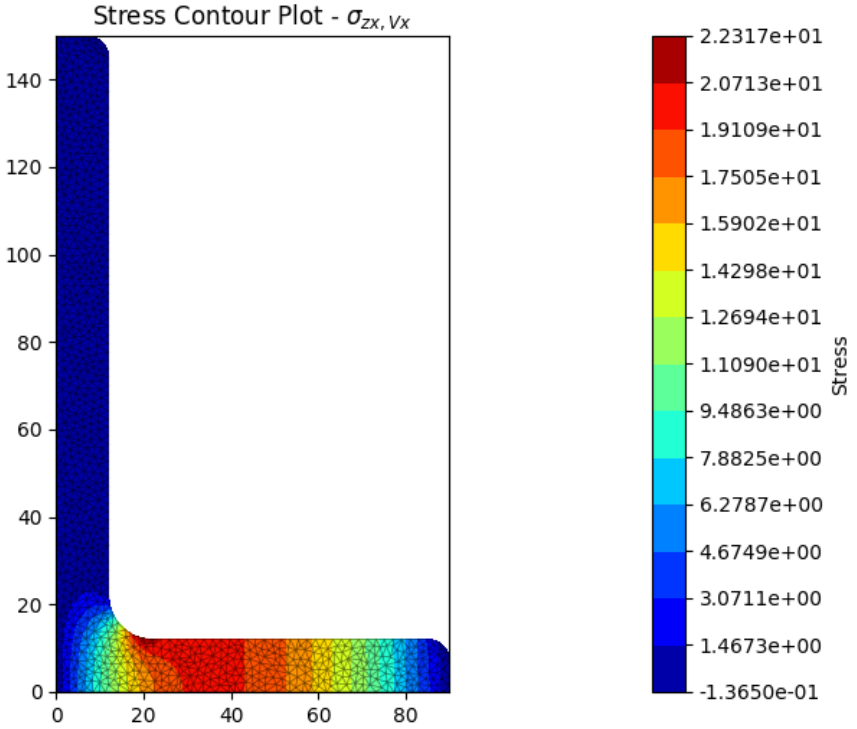


Fig. 103: Contour plot of the shear stress.

plot_stress_vx_zxy(*pause=True*)

Produces a contour plot of the resultant shear stress $\sigma_{zxy,Vx}$ resulting from the shear force V_x.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots a contour of the resultant shear stress within a 150x90x12 UA section resulting from a shear force in the x-direction of 15 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3)

stress_post.plot_stress_vx_zxy()
```

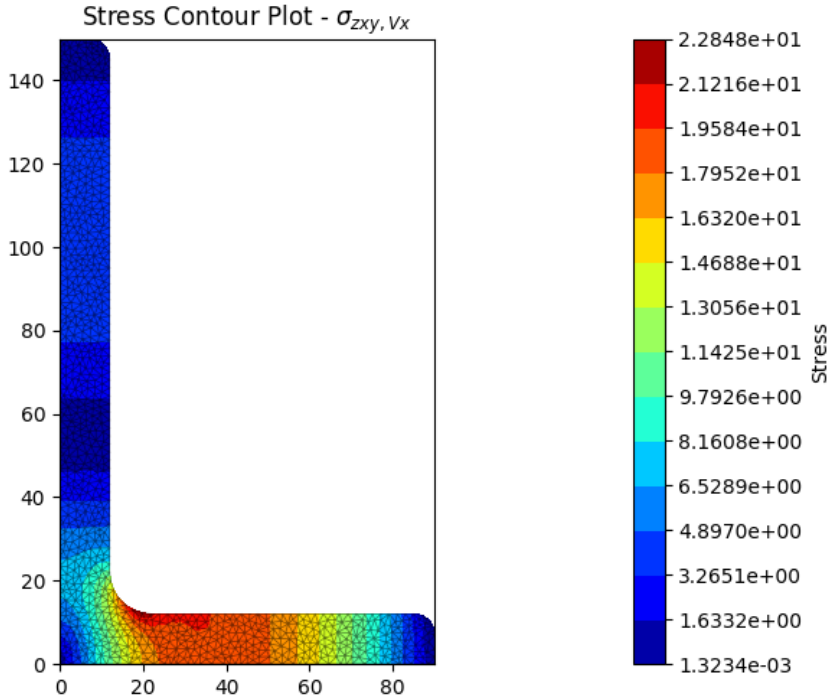


Fig. 104: Contour plot of the shear stress.

plot_stress_vx_zy(*pause=True*)

Produces a contour plot of the *y*-component of the shear stress $\sigma_{zy,Vx}$ resulting from the shear force V_x.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the y-component of the shear stress within a 150x90x12 UA section resulting from a shear force in the x-direction of 15 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3)

stress_post.plot_stress_vx_zy()
```

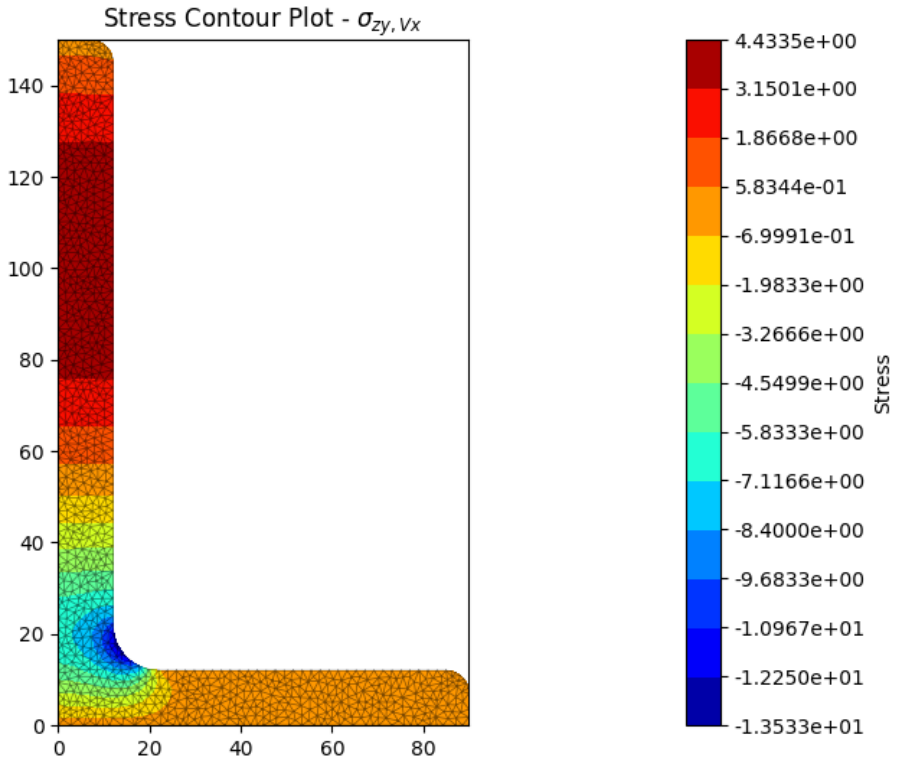


Fig. 105: Contour plot of the shear stress.

plot_stress_vy_zx(*pause=True*)

Produces a contour plot of the x-component of the shear stress $\sigma_{zx,Vy}$ resulting from the shear force V_y.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the x-component of the shear stress within a 150x90x12 UA section resulting from a shear force in the y-direction of 30 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vy=30e3)

stress_post.plot_stress_vy_zx()
```

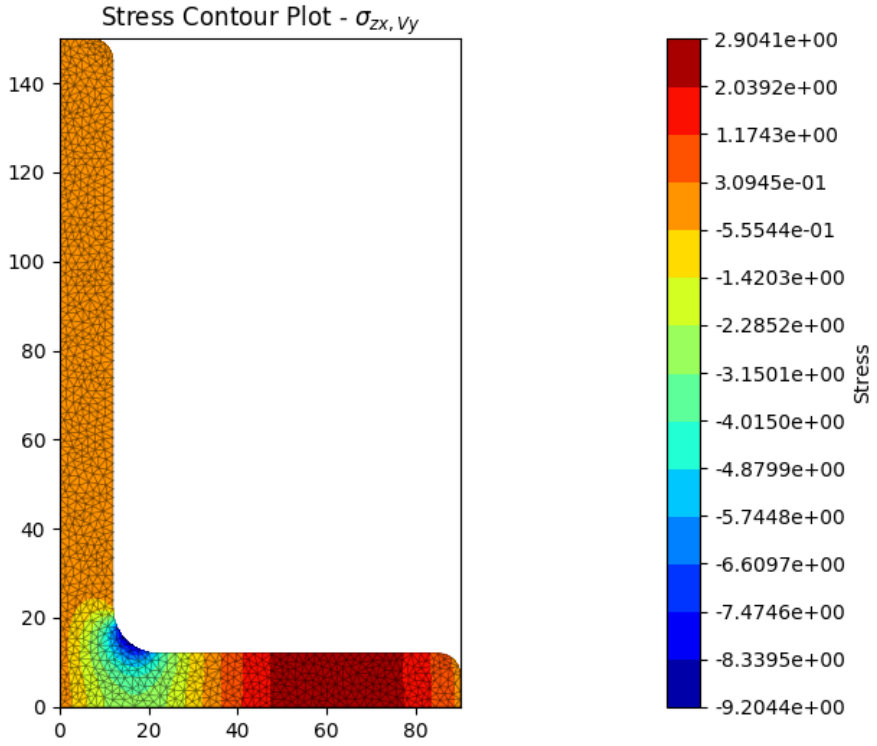


Fig. 106: Contour plot of the shear stress.

plot_stress_vy_zxy(*pause=True*)

Produces a contour plot of the resultant shear stress $\sigma_{zxy,Vy}$ resulting from the shear force V_y.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots a contour of the resultant shear stress within a 150x90x12 UA section resulting from a shear force in the y-direction of 30 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection
```

(continues on next page)

(continued from previous page)

```
geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vy=30e3)

stress_post.plot_stress_vy_zxy()
```

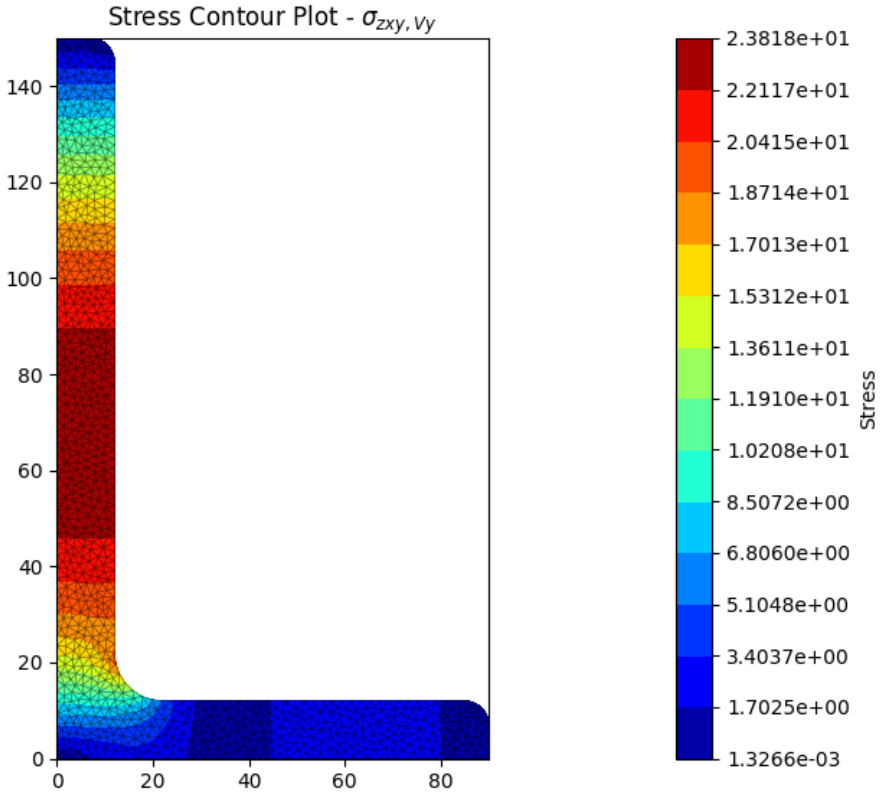


Fig. 107: Contour plot of the shear stress.

plot_stress_vy_zy (*pause=True*)

Produces a contour plot of the *y*-component of the shear stress $\sigma_{zy,Vy}$ resulting from the shear force V_y.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the y-component of the shear stress within a 150x90x12 UA section resulting from a shear force in the y-direction of 30 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection
```

(continues on next page)

(continued from previous page)

```
geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vy=30e3)

stress_post.plot_stress_vy_zy()
```

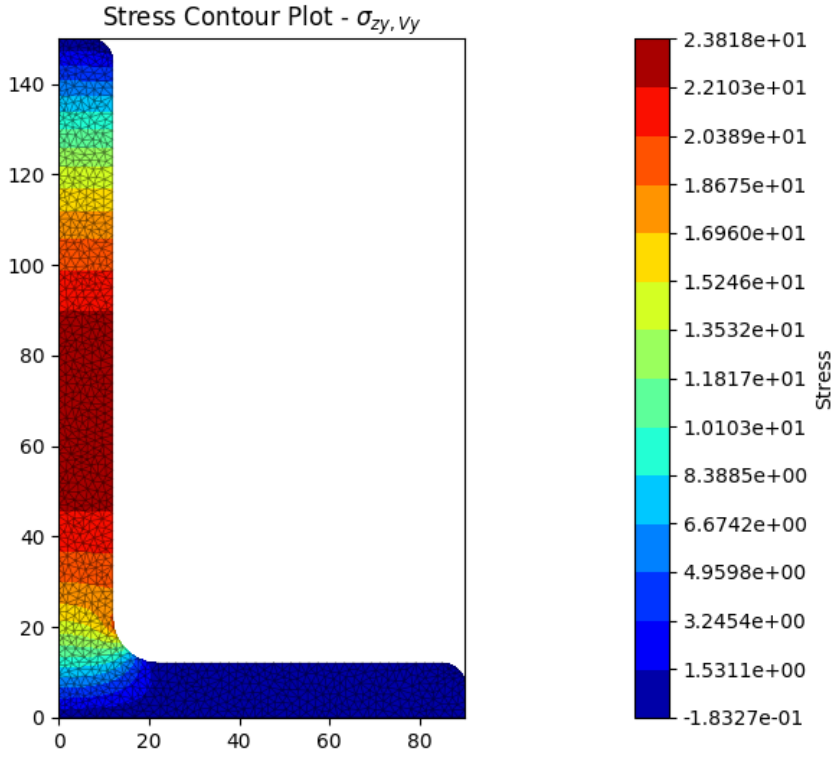


Fig. 108: Contour plot of the shear stress.

plot_stress_zx(*pause=True*)

Produces a contour plot of the x-component of the shear stress σ_{zx} resulting from all actions.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the x-component of the shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m and a shear force of 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
```

(continues on next page)

(continued from previous page)

```
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6, Vy=30e3)

stress_post.plot_stress_zx()
```

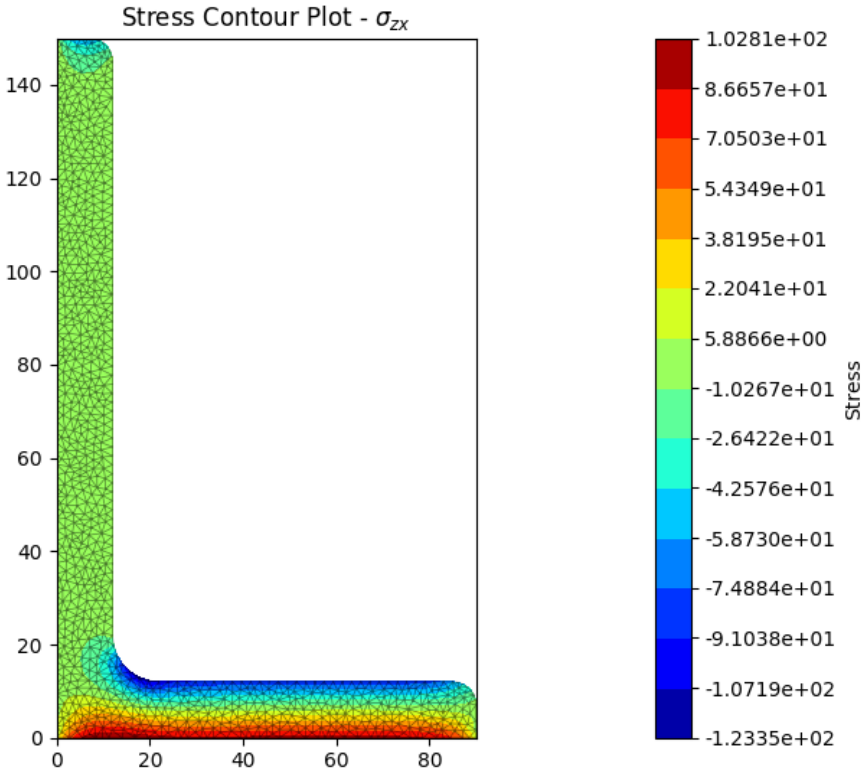


Fig. 109: Contour plot of the shear stress.

plot_stress_zxy(*pause=True*)

Produces a contour plot of the resultant shear stress σ_{zxy} resulting from all actions.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots a contour of the resultant shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m and a shear force of 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)
```

(continues on next page)

(continued from previous page)

```
section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6, Vy=30e3)

stress_post.plot_stress_zxy()
```

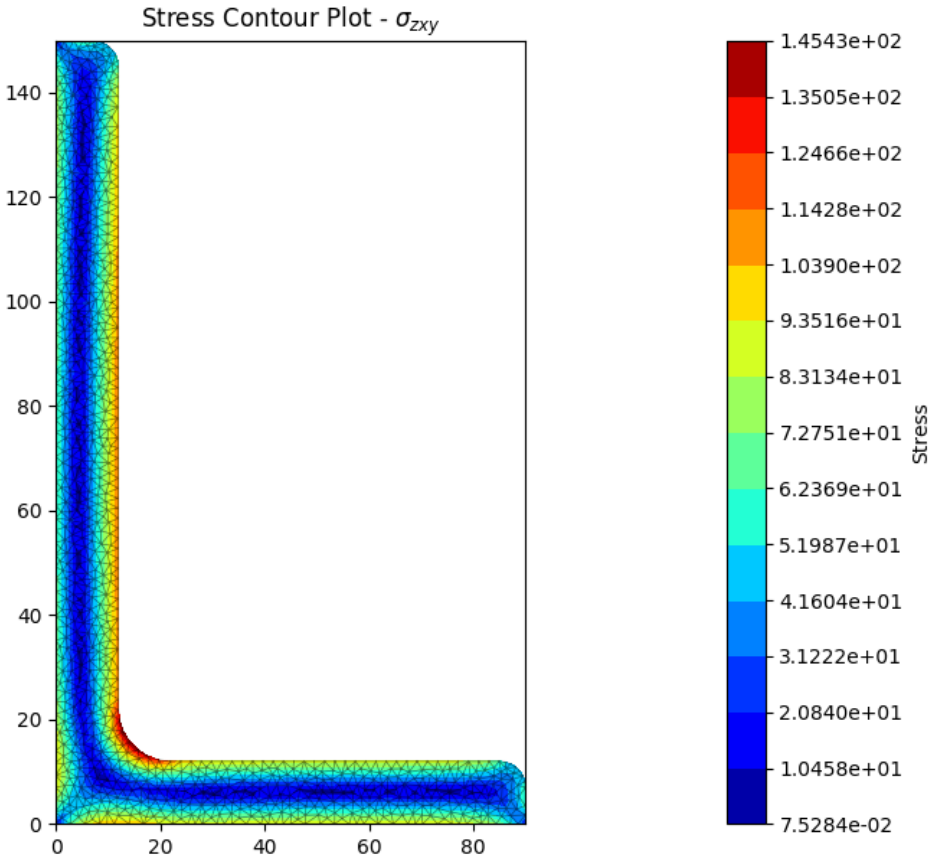


Fig. 110: Contour plot of the shear stress.

plot_stress_zy (*pause=True*)

Produces a contour plot of the y-component of the shear stress σ_{zy} resulting from all actions.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the y-component of the shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m and a shear force of 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
```

(continues on next page)

(continued from previous page)

```
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6, Vy=30e3)

stress_post.plot_stress_zy()
```

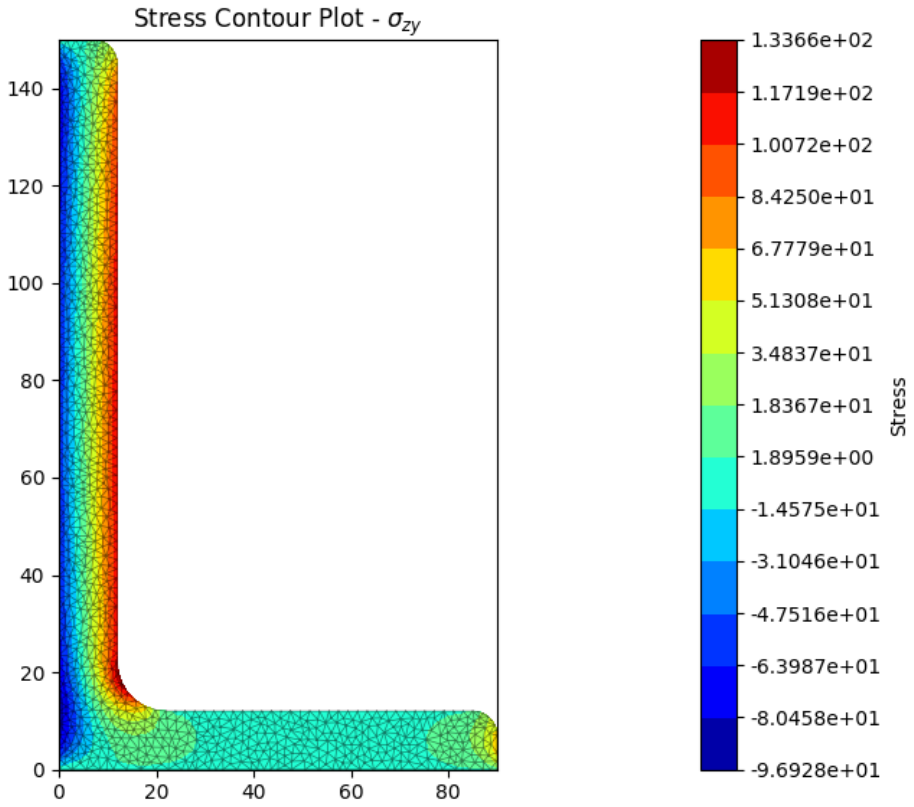


Fig. 111: Contour plot of the shear stress.

plot_stress_zz (*pause=True*)

Produces a contour plot of the combined normal stress σ_{zz} resulting from all actions.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example plots the normal stress within a 150x90x12 UA section resulting from an axial force of 100 kN, a bending moment about the x-axis of 5 kN.m and a bending moment about the y-axis of 2 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
```

(continues on next page)

(continued from previous page)

```
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(N=100e3, Mxx=5e6, Myy=2e6)

stress_post.plot_stress_zz()
```

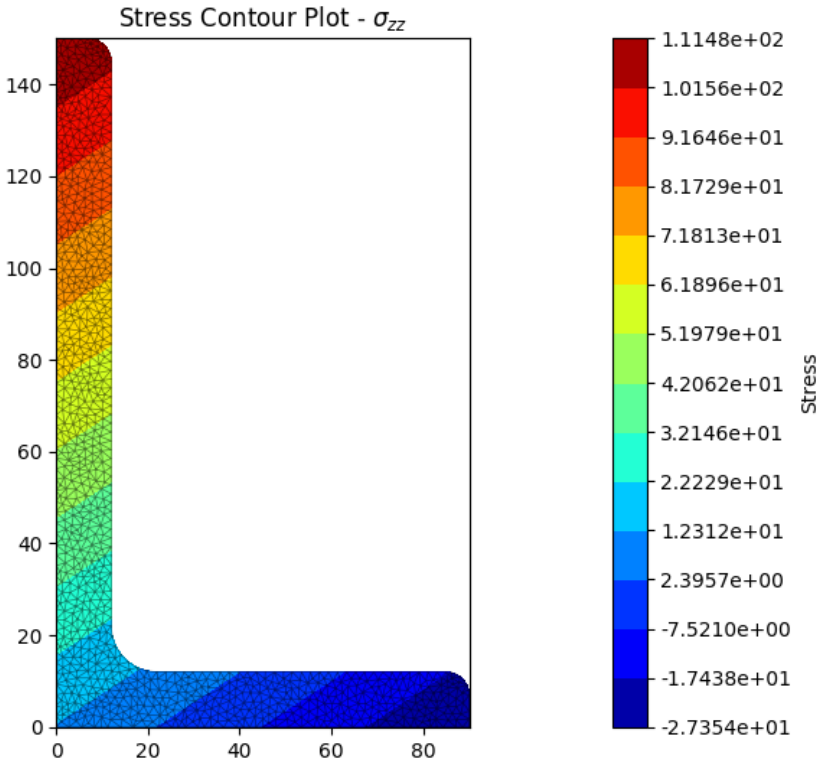


Fig. 112: Contour plot of the normal stress.

plot_vector_mzz_zxy (*pause=True*)

Produces a vector plot of the resultant shear stress $\sigma_{zxy,Mzz}$ resulting from the torsion moment M_{zz}.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example generates a vector plot of the shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)
```

(continues on next page)

(continued from previous page)

```
section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6)

stress_post.plot_vector_mzz_zxy()
```

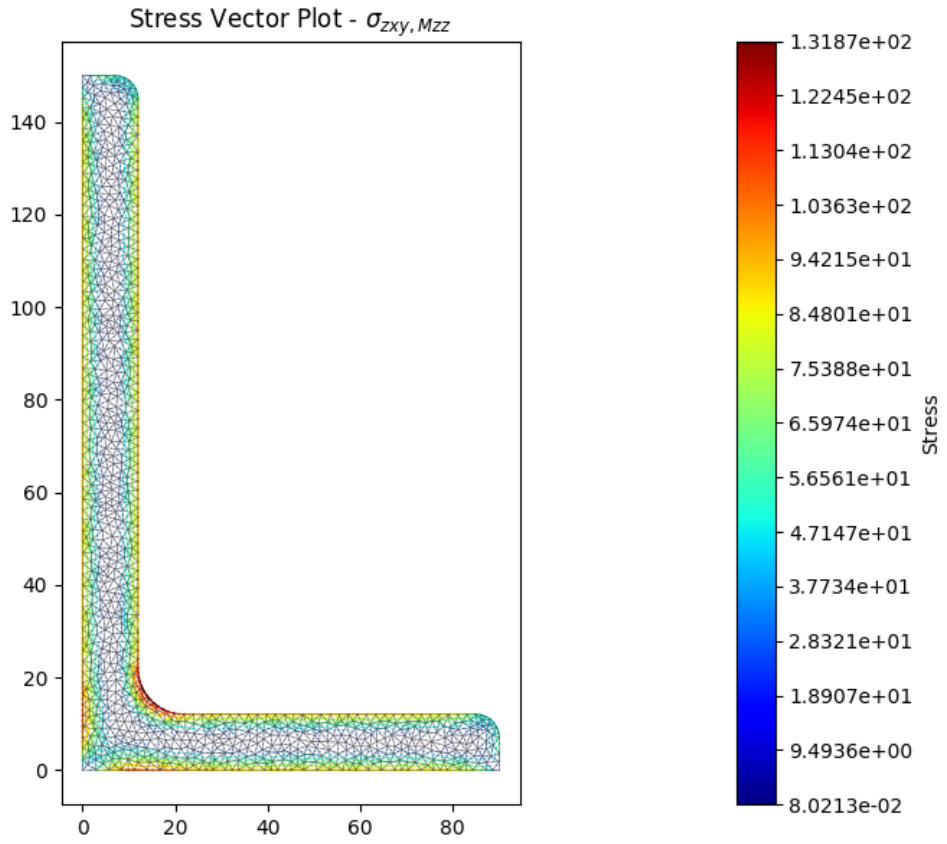


Fig. 113: Vector plot of the shear stress.

plot_vector_v_zxy(*pause=True*)

Produces a vector plot of the resultant shear stress $\sigma_{zxy,\Sigma V}$ resulting from the sum of the applied shear forces $V_x + V_y$.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example generates a vector plot of the shear stress within a 150x90x12 UA section resulting from a shear force of 15 kN in the x-direction and 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
```

(continues on next page)

(continued from previous page)

```
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3, Vy=30e3)

stress_post.plot_vector_v_zxy()
```

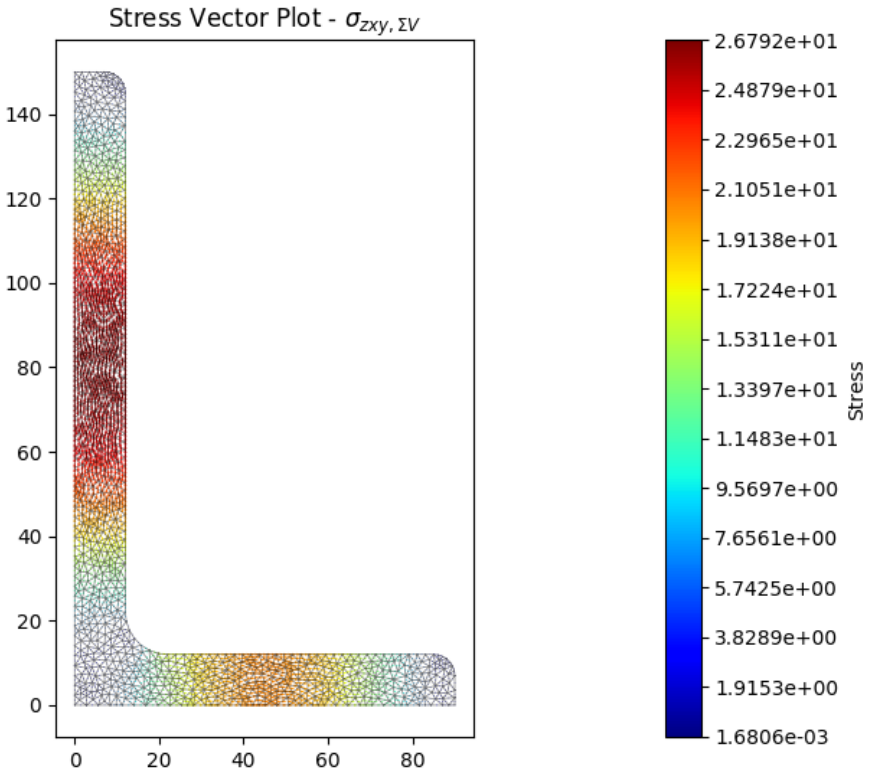


Fig. 114: Vector plot of the shear stress.

plot_vector_vx_zxy (*pause=True*)

Produces a vector plot of the resultant shear stress $\sigma_{zxy,Vx}$ resulting from the shear force V_x.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example generates a vector plot of the shear stress within a 150x90x12 UA section resulting from a shear force in the x-direction of 15 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
```

(continues on next page)

(continued from previous page)

```
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vx=15e3)

stress_post.plot_vector_vx_zxy()
```

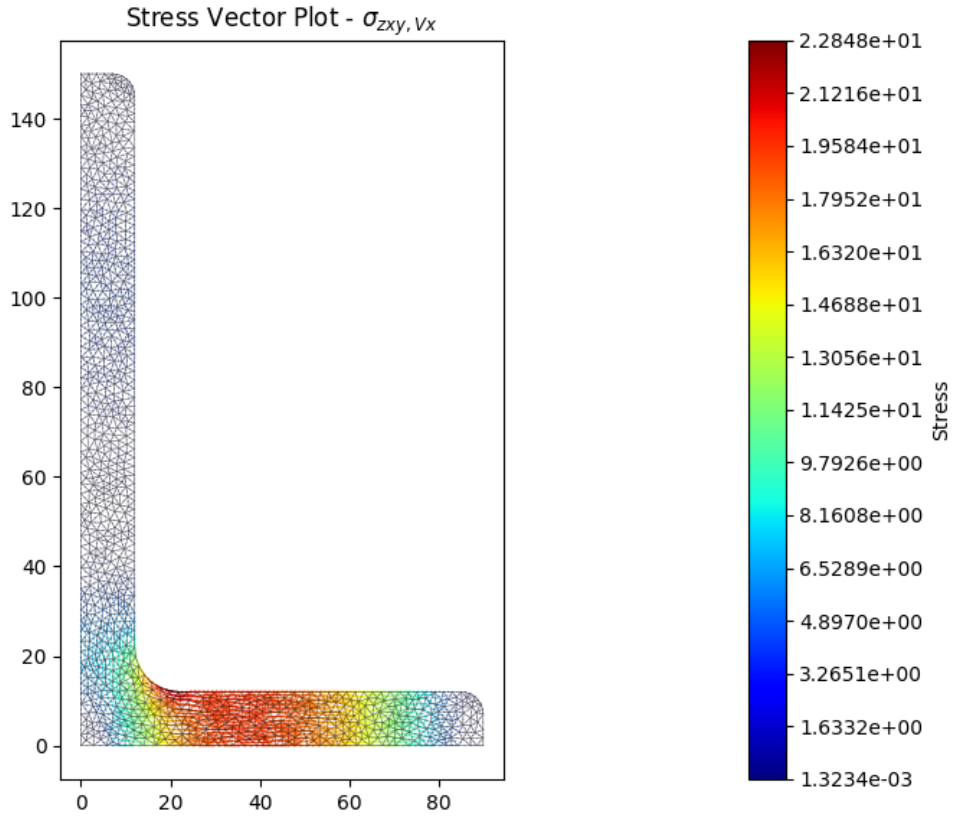


Fig. 115: Vector plot of the shear stress.

plot_vector_vy_zxy(*pause=True*)

Produces a vector plot of the resultant shear stress $\sigma_{zxy,Vy}$ resulting from the shear force V_y.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example generates a vector plot of the shear stress within a 150x90x12 UA section resulting from a shear force in the y-direction of 30 kN:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
```

(continues on next page)

(continued from previous page)

```
section = CrossSection(geometry, mesh)

section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Vy=30e3)

stress_post.plot_vector_vy_zxy()
```

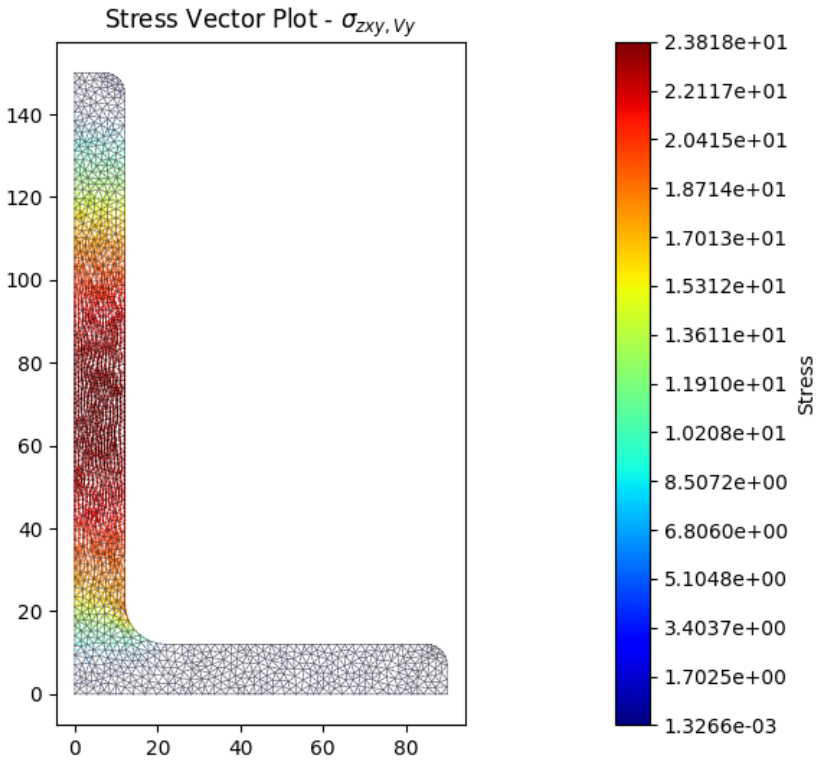


Fig. 116: Vector plot of the shear stress.

plot_vector_zxy(*pause=True*)

Produces a vector plot of the resultant shear stress σ_{zxy} resulting from all actions.

Parameters **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

Returns Matplotlib figure and axes objects (fig, ax)

Return type (matplotlib.figure.Figure, matplotlib.axes)

The following example generates a vector plot of the shear stress within a 150x90x12 UA section resulting from a torsion moment of 1 kN.m and a shear force of 30 kN in the y-direction:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

geometry = sections.AngleSection(d=150, b=90, t=12, r_r=10, r_t=5, n_r=8)
mesh = geometry.create_mesh(mesh_sizes=[2.5])
section = CrossSection(geometry, mesh)
```

(continues on next page)

(continued from previous page)

```
section.calculate_geometric_properties()
section.calculate_warping_properties()
stress_post = section.calculate_stress(Mzz=1e6, Vy=30e3)

stress_post.plot_vector_zxy()
```

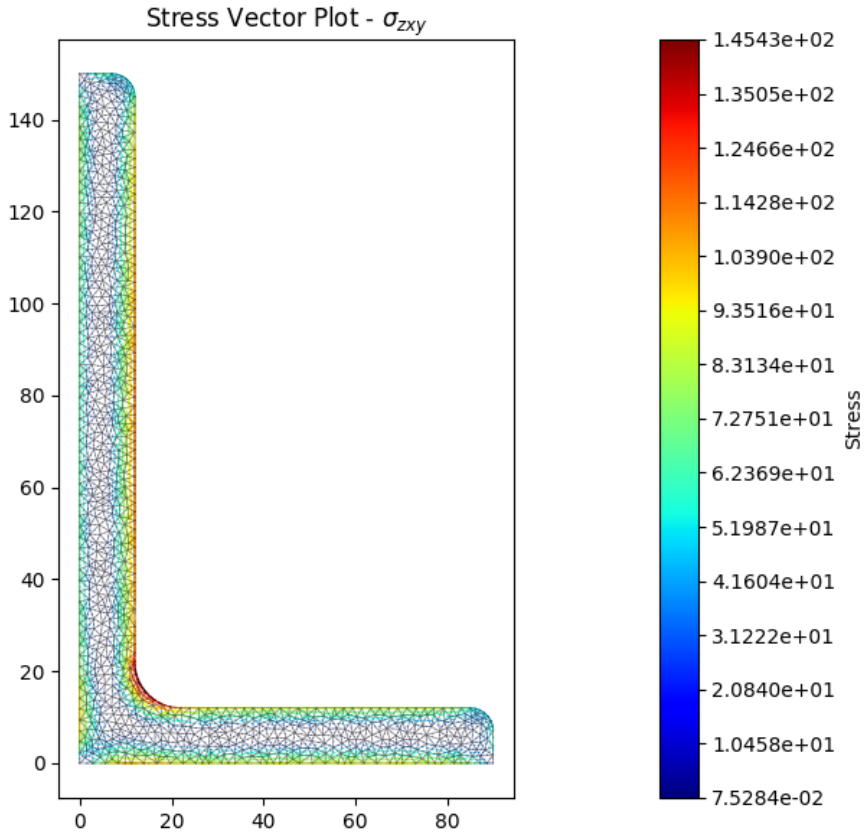


Fig. 117: Vector plot of the shear stress.

MaterialGroup Class

class sectionproperties.analysis.cross_section.**MaterialGroup**(*material*, *num_nodes*)

Bases: object

Class for storing elements of different materials.

A MaterialGroup object contains the finite element objects for a specified *material*. The *stress_result* variable provides storage for stresses related each material.

Parameters

- **material** (*Material*) – Material object for the current MaterialGroup
- **num_nods** (*int*) – Number of nodes for the entire cross-section

Variables

- **material** (*Material*) – Material object for the current MaterialGroup
- **stress_result** (*StressResult*) – A StressResult object for saving the stresses of the current material
- **elements** (list[*Tri6*]) – A list of finite element objects that are of the current material type
- **el_ids** (*list[int]*) – A list of the element IDs of the elements that are of the current material type

add_element(*element*)

Adds an element and its element ID to the MaterialGroup.

Parameters **element** (*Tri6*) – Element to add to the MaterialGroup

StressResult Class

class sectionproperties.analysis.cross_section.**StressResult**(*num_nodes*)

Bases: object

Class for storing a stress result.

Provides variables to store the results from a cross-section stress analysis. Also provides a method to calculate combined stresses.

Parameters **num_nodes** (*int*) – Number of nodes in the finite element mesh

Variables

- **sig_zz_n** (numpy.ndarray) – Normal stress ($\sigma_{zz,N}$) resulting from an axial force
- **sig_zz_mxx** (numpy.ndarray) – Normal stress ($\sigma_{zz,Mxx}$) resulting from a bending moment about the xx-axis
- **sig_zz_myy** (numpy.ndarray) – Normal stress ($\sigma_{zz,Myy}$) resulting from a bending moment about the yy-axis
- **sig_zz_m11** (numpy.ndarray) – Normal stress ($\sigma_{zz,M11}$) resulting from a bending moment about the 11-axis
- **sig_zz_m22** (numpy.ndarray) – Normal stress ($\sigma_{zz,M22}$) resulting from a bending moment about the 22-axis
- **sig_zx_mzz** (numpy.ndarray) – Shear stress ($\sigma_{zx,Mzz}$) resulting from a torsion moment about the zz-axis
- **sig_zy_mzz** (numpy.ndarray) – Shear stress ($\sigma_{zy,Mzz}$) resulting from a torsion moment about the zz-axis
- **sig_zx_vx** (numpy.ndarray) – Shear stress ($\sigma_{zx,Vx}$) resulting from a shear force in the x-direction
- **sig_zy_vx** (numpy.ndarray) – Shear stress ($\sigma_{zy,Vx}$) resulting from a shear force in the x-direction
- **sig_zx_vy** (numpy.ndarray) – Shear stress ($\sigma_{zx,Vy}$) resulting from a shear force in the y-direction
- **sig_zy_vy** (numpy.ndarray) – Shear stress ($\sigma_{zy,Vy}$) resulting from a shear force in the y-direction
- **sig_zz_m** (numpy.ndarray) – Normal stress ($\sigma_{zz,\Sigma M}$) resulting from all bending moments

- **sig_zxy_mzz** (numpy.ndarray) – Resultant shear stress ($\sigma_{zxy,Mzz}$) resulting from a torsion moment in the zz-direction
- **sig_zxy_vx** (numpy.ndarray) – Resultant shear stress ($\sigma_{zxy,Vx}$) resulting from a a shear force in the x-direction
- **sig_zxy_vy** (numpy.ndarray) – Resultant shear stress ($\sigma_{zxy,Vy}$) resulting from a a shear force in the y-direction
- **sig_zx_v** (numpy.ndarray) – Shear stress ($\sigma_{zx,\Sigma V}$) resulting from all shear forces
- **sig_zy_v** (numpy.ndarray) – Shear stress ($\sigma_{zy,\Sigma V}$) resulting from all shear forces
- **sig_zxy_v** (numpy.ndarray) – Resultant shear stress ($\sigma_{zxy,\Sigma V}$) resulting from all shear forces
- **sig_zz** (numpy.ndarray) – Combined normal force (σ_{zz}) resulting from all actions
- **sig_zx** (numpy.ndarray) – Combined shear stress (σ_{zx}) resulting from all actions
- **sig_zy** (numpy.ndarray) – Combined shear stress (σ_{zy}) resulting from all actions
- **sig_zxy** (numpy.ndarray) – Combined resultant shear stress (σ_{zxy}) resulting from all actions
- **sig_vm** (numpy.ndarray) – von Mises stress (σ_{VM}) resulting from all actions

calculate_combined_stresses()
Calculates the combined cross-section stresses.

SectionProperties Class

class sectionproperties.analysis.cross_section.**SectionProperties**
Bases: object

Class for storing section properties.

Stores calculated section properties. Also provides methods to calculate section properties entirely derived from other section properties.

Variables

- **area** (*float*) – Cross-sectional area
- **perimeter** (*float*) – Cross-sectional perimeter
- **ea** (*float*) – Modulus weighted area (axial rigidity)
- **ga** (*float*) – Modulus weighted product of shear modulus and area
- **nu_eff** (*float*) – Effective Poisson's ratio
- **qx** (*float*) – First moment of area about the x-axis
- **qy** (*float*) – First moment of area about the y-axis
- **ixx_g** (*float*) – Second moment of area about the global x-axis
- **iyy_g** (*float*) – Second moment of area about the global y-axis
- **ixy_g** (*float*) – Second moment of area about the global xy-axis
- **cx** (*float*) – X coordinate of the elastic centroid
- **cy** (*float*) – Y coordinate of the elastic centroid
- **ixx_c** (*float*) – Second moment of area about the centroidal x-axis

- **iyy_c** (*float*) – Second moment of area about the centroidal y-axis
- **ixy_c** (*float*) – Second moment of area about the centroidal xy-axis
- **zxx_plus** (*float*) – Section modulus about the centroidal x-axis for stresses at the positive extreme value of y
- **zxx_minus** (*float*) – Section modulus about the centroidal x-axis for stresses at the negative extreme value of y
- **zyy_plus** (*float*) – Section modulus about the centroidal y-axis for stresses at the positive extreme value of x
- **zyy_minus** (*float*) – Section modulus about the centroidal y-axis for stresses at the negative extreme value of x
- **rx_c** (*float*) – Radius of gyration about the centroidal x-axis.
- **ry_c** (*float*) – Radius of gyration about the centroidal y-axis.
- **i11_c** (*float*) – Second moment of area about the centroidal 11-axis
- **i22_c** (*float*) – Second moment of area about the centroidal 22-axis
- **phi** (*float*) – Principal axis angle
- **z11_plus** (*float*) – Section modulus about the principal 11-axis for stresses at the positive extreme value of the 22-axis
- **z11_minus** (*float*) – Section modulus about the principal 11-axis for stresses at the negative extreme value of the 22-axis
- **z22_plus** (*float*) – Section modulus about the principal 22-axis for stresses at the positive extreme value of the 11-axis
- **z22_minus** (*float*) – Section modulus about the principal 22-axis for stresses at the negative extreme value of the 11-axis
- **r11_c** (*float*) – Radius of gyration about the principal 11-axis.
- **r22_c** (*float*) – Radius of gyration about the principal 22-axis.
- **j** (*float*) – Torsion constant
- **omega** (numpy.ndarray) – Warping function
- **psi_shear** (numpy.ndarray) – Psi shear function
- **phi_shear** (numpy.ndarray) – Phi shear function
- **Delta_s** (*float*) – Shear factor
- **x_se** (*float*) – X coordinate of the shear centre (elasticity approach)
- **y_se** (*float*) – Y coordinate of the shear centre (elasticity approach)
- **x11_se** (*float*) – 11 coordinate of the shear centre (elasticity approach)
- **y22_se** (*float*) – 22 coordinate of the shear centre (elasticity approach)
- **x_st** (*float*) – X coordinate of the shear centre (Trefftz's approach)
- **y_st** (*float*) – Y coordinate of the shear centre (Trefftz's approach)
- **gamma** (*float*) – Warping constant
- **A_sx** (*float*) – Shear area about the x-axis
- **A_sy** (*float*) – Shear area about the y-axis

- **A_sxy** (*float*) – Shear area about the xy-axis
- **A_s11** (*float*) – Shear area about the 11 bending axis
- **A_s22** (*float*) – Shear area about the 22 bending axis
- **beta_x_plus** (*float*) – Monosymmetry constant for bending about the x-axis with the top flange in compression
- **beta_x_minus** (*float*) – Monosymmetry constant for bending about the x-axis with the bottom flange in compression
- **beta_y_plus** (*float*) – Monosymmetry constant for bending about the y-axis with the top flange in compression
- **beta_y_minus** (*float*) – Monosymmetry constant for bending about the y-axis with the bottom flange in compression
- **beta_11_plus** (*float*) – Monosymmetry constant for bending about the 11-axis with the top flange in compression
- **beta_11_minus** (*float*) – Monosymmetry constant for bending about the 11-axis with the bottom flange in compression
- **beta_22_plus** (*float*) – Monosymmetry constant for bending about the 22-axis with the top flange in compression
- **beta_22_minus** (*float*) – Monosymmetry constant for bending about the 22-axis with the bottom flange in compression
- **x_pc** (*float*) – X coordinate of the global plastic centroid
- **y_pc** (*float*) – Y coordinate of the global plastic centroid
- **x11_pc** (*float*) – 11 coordinate of the principal plastic centroid
- **y22_pc** (*float*) – 22 coordinate of the principal plastic centroid
- **sxx** (*float*) – Plastic section modulus about the centroidal x-axis
- **syy** (*float*) – Plastic section modulus about the centroidal y-axis
- **sf_xx_plus** (*float*) – Shape factor for bending about the x-axis with respect to the top fibre
- **sf_xx_minus** (*float*) – Shape factor for bending about the x-axis with respect to the bottom fibre
- **sf_yy_plus** (*float*) – Shape factor for bending about the y-axis with respect to the top fibre
- **sf_yy_minus** (*float*) – Shape factor for bending about the y-axis with respect to the bottom fibre
- **s11** (*float*) – Plastic section modulus about the 11-axis
- **s22** (*float*) – Plastic section modulus about the 22-axis
- **sf_11_plus** (*float*) – Shape factor for bending about the 11-axis with respect to the top fibre
- **sf_11_minus** (*float*) – Shape factor for bending about the 11-axis with respect to the bottom fibre
- **sf_22_plus** (*float*) – Shape factor for bending about the 22-axis with respect to the top fibre

- **sf_22_minus** (*float*) – Shape factor for bending about the 22-axis with respect to the bottom fibre

calculate_centroidal_properties(*mesh*)
Calculates the geometric section properties about the centroidal and principal axes based on the results about the global axis.

calculate_elastic_centroid()
Calculates the elastic centroid based on the cross-section area and first moments of area.

7.2.2 *fea* Module

Tri6 Class

class sectionproperties.analysis.fea.**Tri6**(*el_id*, *coords*, *node_ids*, *material*)
Bases: object

Class for a six noded quadratic triangular element.

Provides methods for the calculation of section properties based on the finite element method.

Parameters

- **el_id** (*int*) – Unique element id
- **coords** (numpy.ndarray) – A 2 x 6 array of the coordinates of the tri-6 nodes. The first three columns relate to the vertices of the triangle and the last three columns correspond to the mid-nodes.
- **node_ids** (*list[int]*) – A list of the global node ids for the current element
- **material** (*Material*) – Material object for the current finite element.

Variables

- **el_id** (*int*) – Unique element id
- **coords** (numpy.ndarray) – A 2 x 6 array of the coordinates of the tri-6 nodes. The first three columns relate to the vertices of the triangle and the last three columns correspond to the mid-nodes.
- **node_ids** (*list[int]*) – A list of the global node ids for the current element
- **material** (*Material*) – Material of the current finite element.

element_stress(*N*, *Mxx*, *Myy*, *M11*, *M22*, *Mzz*, *Vx*, *Vy*, *ea*, *cx*, *cy*, *ixx*, *iyy*, *ixy*, *i11*, *i22*, *phi*, *j*, *nu*, *omega*, *psi_shear*, *phi_shear*, *Delta_s*)
Calculates the stress within an element resulting from a specified loading. Also returns the shape function weights.

Parameters

- **N** (*float*) – Axial force
- **Mxx** (*float*) – Bending moment about the centroidal xx-axis
- **Myy** (*float*) – Bending moment about the centroidal yy-axis
- **M11** (*float*) – Bending moment about the centroidal 11-axis
- **M22** (*float*) – Bending moment about the centroidal 22-axis
- **Mzz** (*float*) – Torsion moment about the centroidal zz-axis
- **Vx** (*float*) – Shear force acting in the x-direction

- **Vy** (*float*) – Shear force acting in the y-direction
- **ea** (*float*) – Modulus weighted area
- **cx** (*float*) – x position of the elastic centroid
- **cy** (*float*) – y position of the elastic centroid
- **ixx** (*float*) – Second moment of area about the centroidal x-axis
- **iyy** (*float*) – Second moment of area about the centroidal y-axis
- **ixy** (*float*) – Second moment of area about the centroidal xy-axis
- **i11** (*float*) – Second moment of area about the principal 11-axis
- **i22** (*float*) – Second moment of area about the principal 22-axis
- **phi** (*float*) – Principal bending axis angle
- **j** (*float*) – St. Venant torsion constant
- **nu** (*float*) – Effective Poisson's ratio for the cross-section
- **omega** (numpy.ndarray) – Values of the warping function at the element nodes
- **psi_shear** (numpy.ndarray) – Values of the psi shear function at the element nodes
- **phi_shear** (numpy.ndarray) – Values of the phi shear function at the element nodes
- **Delta_s** (*float*) – Cross-section shear factor

Returns Tuple containing element stresses and integration weights ($\sigma_{zz,n}$, $\sigma_{zz,mxx}$, $\sigma_{zz,myy}$, $\sigma_{zz,m11}$, $\sigma_{zz,m22}$, $\sigma_{zx,mzz}$, $\sigma_{zy,mzz}$, $\sigma_{zx,vx}$, $\sigma_{zy,vx}$, $\sigma_{zx,vy}$, $\sigma_{zy,vy}$, w_i)

Return type tuple(numpy.ndarray, numpy.ndarray, ...)

geometric_properties()

Calculates the geometric properties for the current finite element.

Returns Tuple containing the geometric properties and the elastic and shear moduli of the element: *(area, qx, qy, ixx, iyy, ixy, e, g)*

Return type tuple(float)

monosymmetry_integrals(*phi*)

Calculates the integrals used to evaluate the monosymmetry constant about both global axes and both prinicipal axes.

Parameters **phi** (*float*) – Principal bending axis angle

Returns Integrals used to evaluate the monosymmetry constants *(int_x, int_y, int_11, int_22)*

Return type tuple(float, float, float, float)

plastic_properties(*u*, *p*)

Calculates total force resisted by the element when subjected to a stress equal to the yield strength. Also returns the modulus weighted area and first moments of area, and determines whether or not the element is above or below the line defined by the unit vector *u* and point *p*.

Parameters

- **u** (numpy.ndarray) – Unit vector in the direction of the line
- **p** (numpy.ndarray) – Point on the line

Returns Element force *(force)*, modulus weighted area properties *(ea, e.qx, e.qy)* and whether or not the element is above the line

Return type tuple(float, float, float, float, bool)

point_within_element(*pt*)
Determines whether a point lies within the current element.

Parameters **pt** (*list[float, float]*) – Point to check *(x, y)*

Returns Whether the point lies within an element

Return type bool

shear_coefficients(*ixx*, *iyy*, *ixy*, *psi_shear*, *phi_shear*, *nu*)
Calculates the variables used to determine the shear deformation coefficients.

Parameters

- **ixx** (*float*) – Second moment of area about the centroidal x-axis
- **iyy** (*float*) – Second moment of area about the centroidal y-axis
- **ixy** (*float*) – Second moment of area about the centroidal xy-axis
- **psi_shear** (numpy.ndarray) – Values of the psi shear function at the element nodes
- **phi_shear** (numpy.ndarray) – Values of the phi shear function at the element nodes
- **nu** (*float*) – Effective Poisson's ratio for the cross-section

Returns Shear deformation variables *(kappa_x, kappa_y, kappa_xy)*

Return type tuple(float, float, float)

shear_load_vectors(*ixx*, *iyy*, *ixy*, *nu*)
Calculates the element shear load vectors used to evaluate the shear functions.

Parameters

- **ixx** (*float*) – Second moment of area about the centroidal x-axis
- **iyy** (*float*) – Second moment of area about the centroidal y-axis
- **ixy** (*float*) – Second moment of area about the centroidal xy-axis
- **nu** (*float*) – Effective Poisson's ratio for the cross-section

Returns Element shear load vector psi *(f_psi)* and phi *(f_phi)*

Return type tuple(numpy.ndarray, numpy.ndarray)

shear_warping_integrals(*ixx*, *iyy*, *ixy*, *omega*)
Calculates the element shear centre and warping integrals required for shear analysis of the cross-section.

Parameters

- **ixx** (*float*) – Second moment of area about the centroidal x-axis
- **iyy** (*float*) – Second moment of area about the centroidal y-axis
- **ixy** (*float*) – Second moment of area about the centroidal xy-axis
- **omega** (numpy.ndarray) – Values of the warping function at the element nodes

Returns Shear centre integrals about the x and y-axes *(sc_xint, sc_yint)*, warping integrals *(q_omega, i_omega, i_xomega, i_yomega)*

Return type tuple(float, float, float, float, float, float)

torsion_properties()
Calculates the element stiffness matrix used for warping analysis and the torsion load vector.

Returns Element stiffness matrix *(k_el)* and element torsion load vector *(f_el)*

Return type tuple(numpy.ndarray, numpy.ndarray)

fea Functions

sectionproperties.analysis.fea.**gauss_points**(*n*)
Returns the Gaussian weights and locations for *n* point Gaussian integration of a quadratic triangular element.

Parameters **n** (*int*) – Number of Gauss points (1, 3 or 6)

Returns An *n x 4* matrix consisting of the integration weight and the eta, xi and zeta locations for *n* Gauss points

Return type numpy.ndarray

sectionproperties.analysis.fea.**shape_function**(*coords*, *gauss_point*)
Computes shape functions, shape function derivatives and the determinant of the Jacobian matrix for a tri 6 element at a given Gauss point.

Parameters

- **coords** (numpy.ndarray) – Global coordinates of the quadratic triangle vertices [2 x 6]
- **gauss_point** (numpy.ndarray) – Gaussian weight and isoparametric location of the Gauss point

Returns The value of the shape functions *N(i)* at the given Gauss point [1 x 6], the derivative of the shape functions in the j-th global direction *B(i,j)* [2 x 6] and the determinant of the Jacobian matrix *j*

Return type tuple(numpy.ndarray, numpy.ndarray, float)

sectionproperties.analysis.fea.**extrapolate_to_nodes**(*w*)
Extrapolates results at six Gauss points to the six noes of a quadratic triangular element.

Parameters **w** (numpy.ndarray) – Result at the six Gauss points [1 x 6]

Returns Extrapolated nodal values at the six nodes [1 x 6]

Return type numpy.ndarray

sectionproperties.analysis.fea.**principal_coordinate**(*phi*, *x*, *y*)
Determines the coordinates of the cartesian point *(x, y)* in the principal axis system given an axis rotation angle phi.

Parameters

- **phi** (*float*) – Prinicpal bending axis angle (degrees)
- **x** (*float*) – x coordinate in the global axis
- **y** (*float*) – y coordinate in the global axis

Returns Principal axis coordinates *(x1, y2)*

Return type tuple(float, float)

sectionproperties.analysis.fea.**global_coordinate**(*phi*, *x11*, *y22*)
Determines the global coordinates of the principal axis point *(x1, y2)* given principal axis rotation angle phi.

Parameters

- **phi** (*float*) – Prinicpal bending axis angle (degrees)

- **x11** (*float*) – 11 coordinate in the principal axis
- **y22** (*float*) – 22 coordinate in the principal axis

Returns Global axis coordinates *(x, y)*

Return type tuple(float, float)

sectionproperties.analysis.fea.**point_above_line**(*u*, *px*, *py*, *x*, *y*)

Determines whether a point *(x, y)* is a above or below the line defined by the parallel unit vector *u* and the point *(px, py)*.

Parameters

- **u** (numpy.ndarray) – Unit vector parallel to the line [1 x 2]
- **px** (*float*) – x coordinate of a point on the line
- **py** (*float*) – y coordinate of a point on the line
- **x** (*float*) – x coordinate of the point to be tested
- **y** (*float*) – y coordinate of the point to be tested

Returns This method returns *True* if the point is above the line or *False* if the point is below the line

Return type bool

7.2.3 *solver* Module

solver Functions

sectionproperties.analysis.solver.**solve_cgs**(*k*, *f*, *m=None*, *tol=1e-05*)

Solves a linear system of equations (Ku = f) using the CGS iterative method.

Parameters

- **k** (scipy.sparse.csc_matrix) – N x N matrix of the linear system
- **f** (numpy.ndarray) – N x 1 right hand side of the linear system
- **tol** (*float*) – Tolerance for the solver to acheieve. The algorithm terminates when either the relative or the absolute residual is below tol.
- **m** (scipy.linalg.LinearOperator) – Preconditioner for the linear matrix approximating the inverse of k

Returns The solution vector to the linear system of equations

Return type numpy.ndarray

Raises **RuntimeError** – If the CGS iterative method does not converge

sectionproperties.analysis.solver.**solve_cgs_lagrange**(*k_lg*, *f*, *tol=1e-05*, *m=None*)

Solves a linear system of equations (Ku = f) using the CGS iterative method and the Lagrangian multiplier method.

Parameters

- **k** (scipy.sparse.csc_matrix) – (N+1) x (N+1) Lagrangian multiplier matrix of the linear system
- **f** (numpy.ndarray) – N x 1 right hand side of the linear system

- **tol** (*float*) – Tolerance for the solver to acheieve. The algorithm terminates when either the relative or the absolute residual is below tol.
- **m** (scipy.linalg.LinearOperator) – Preconditioner for the linear matrix approximating the inverse of k

Returns The solution vector to the linear system of equations

Return type numpy.ndarray

Raises **RuntimeError** – If the CGS iterative method does not converge or the error from the Lagrangian multiplier method exceeds the tolerance

sectionproperties.analysis.solver.**solve_direct**(*k*, *f*)

Solves a linear system of equations (Ku = f) using the direct solver method.

Parameters

- **k** (scipy.sparse.csc_matrix) – N x N matrix of the linear system
- **f** (numpy.ndarray) – N x 1 right hand side of the linear system

Returns The solution vector to the linear system of equations

Return type numpy.ndarray

sectionproperties.analysis.solver.**solve_direct_lagrange**(*k_lg*, *f*)

Solves a linear system of equations (Ku = f) using the direct solver method and the Lagrangian multiplier method.

Parameters

- **k** (scipy.sparse.csc_matrix) – (N+1) x (N+1) Lagrangian multiplier matrix of the linear system
- **f** (numpy.ndarray) – N x 1 right hand side of the linear system

Returns The solution vector to the linear system of equations

Return type numpy.ndarray

Raises **RuntimeError** – If the Lagrangian multiplier method exceeds a tolerance of 1e-5

sectionproperties.analysis.solver.**function_timer**(*text*, *function*, **args*)

Displays the message *text* and returns the time taken for a function, with arguments *args*, to execute. The value returned by the timed function is also returned.

Parameters

- **text** (*string*) – Message to display
- **function** (*function*) – Function to time and execute
- **args** – Function arguments

Returns Value returned from the function

7.3 Post-Processor Package

7.3.1 *post* Module

post Functions

sectionproperties.post.post.**setup_plot**(*ax*, *pause*)
Executes code required to set up a matplotlib figure.

Parameters

- **ax** (matplotlib.axes.Axes) – Axes object on which to plot
- **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.

sectionproperties.post.post.**finish_plot**(*ax*, *pause*, *title=''*)
Executes code required to finish a matplotlib figure.

Parameters

- **ax** (matplotlib.axes.Axes) – Axes object on which to plot
- **pause** (*bool*) – If set to true, the figure pauses the script until the window is closed. If set to false, the script continues immediately after the window is rendered.
- **title** (*string*) – Plot title

sectionproperties.post.post.**draw_principal_axis**(*ax*, *phi*, *cx*, *cy*)
Draws the principal axis on a plot.

Parameters

- **ax** (matplotlib.axes.Axes) – Axes object on which to plot
- **phi** (*float*) – Principal axis angle in radians
- **cx** (*float*) – x-location of the centroid
- **cy** (*float*) – y-location of the centroid

sectionproperties.post.post.**print_results**(*cross_section*, *fmt*)
Prints the results that have been calculated to the terminal.

Parameters

- **cross_section** (*CrossSection*) – Structural cross-section object
- **fmt** (*string*) – Number format

CHAPTER 8

Theoretical Background

coming soon...

Here's a quick example that harnesses some of the power of *sectionproperties* and shows its simplicity:

```
import sectionproperties.pre.sections as sections
from sectionproperties.analysis.cross_section import CrossSection

# create geometry of the cross-section
geometry = sections.ISection(d=203, b=133, t_f=7.8, t_w=5.8, r=8.9, n_r=8)

# generate a finite element mesh
mesh = geometry.create_mesh(mesh_sizes=[2.5])

# create a CrossSection object for analysis
section = CrossSection(geometry, mesh)

# calculate various cross-section properties
section.calculate_geometric_properties()
section.calculate_warping_properties()

# print some of the calculated section properties
print(section.get_area())  # cross-section area
>>>3231.80
print(section.get_ic())  # second moments of area about the centroidal axis
>>>(23544664.29, 3063383.07, 0.00)
print(section.get_j())  # torsion constant
>>>62907.79
print(section.get_As())  # shear areas in the x & y directions
>>>(1842.17, 1120.18)
```

CHAPTER 9

Support

Contact me on my email robbie.vanleeuwen@gmail.com or raise an issue on the github issue tracker using one of the issue templates. If you have a request for a feature to be added to the *sectionproperties* package, please don't hesitate to get in touch

CHAPTER 10

License

The project is licensed under the MIT license.

Index

D

E

F

G

H

I

L

M

N

O

P

R

S

T

Z