

---

# **Abdbeam Documentation**

***Release 0.2.1***

**Danilo S. Victorazzo**

**Feb 11, 2019**



---

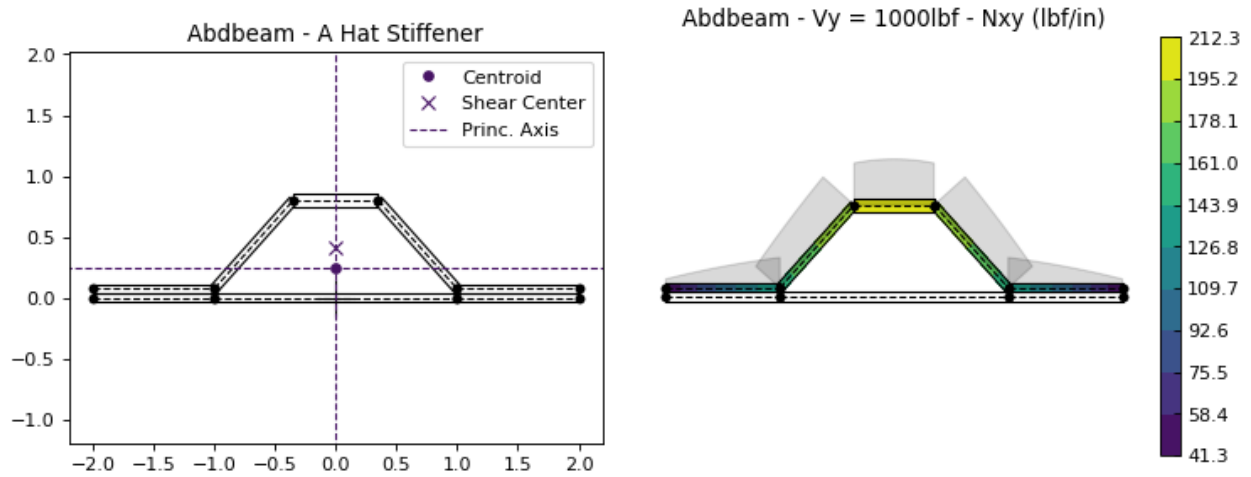
## Contents

---

<b>1</b>	<b>Contents</b>	<b>3</b>
<b>2</b>	<b>Source Code</b>	<b>37</b>
<b>3</b>	<b>Quick Example</b>	<b>39</b>
<b>4</b>	<b>Indices and tables</b>	<b>43</b>
	<b>Python Module Index</b>	<b>45</b>



*Abdbeam* is a python package for the cross section analysis of thin-walled composite material beams of any shape.



These are a few things you can do with *Abdbeam*:

- Use a fast thin-walled anisotropic composite beam theory including closed cells, open branches, shear connectors and booms<sup>1</sup>;
- Recover replacement stiffnesses (EA, EI<sub>yy</sub>, EI<sub>zz</sub>, EI<sub>yz</sub>, GJ) and/or a full 4 x 4 stiffness matrix for beams with arbitrary layups and shapes;
- Recover centroid and shear center locations;
- Obtain internal load distributions (N<sub>x</sub>, N<sub>xy</sub>, M<sub>x</sub>, M<sub>y</sub>, M<sub>xy</sub> for segments; P<sub>x</sub> and T<sub>x</sub> for booms) for a large number of cross section load cases (defined by P<sub>x</sub>, M<sub>y</sub>, M<sub>z</sub>, T<sub>z</sub>, V<sub>y</sub> and V<sub>z</sub> section loads);
- Plot cross sections, their properties and internal loads.

<sup>1</sup> Booms are discrete stiffeners containing axial and torsional stiffnesses.



## 1.1 Installation

### 1.1.1 Windows and UNIX (MacOS/Linux)

*Abdbeam* is installed using PyPI ([Python package index](#)):

```
pip install abdbeam
```

### 1.1.2 Dependencies

The following external packages are required to run *Abdbeam*:

- [Numpy](#)
- [Pandas](#)
- [Matplotlib](#)

## 1.2 Creating Sections

When creating your own arbitrary sections, pay attention to the following modeling rules:

- No floating segments;
- No floating points;
- No segments crossing each other;
- No coincident or partially overlapping segments;
- Unique integer ids for materials, segments and points.

The best place to start creating your own sections is referring to the *Abdbeam* applications found in the [Examples](#) page.

## 1.3 Examples

This page shows application examples with open sections, sections with branches, closed sections, multi-cells, booms and shear connectors. You can also locate these examples as individual modules in the `abdbeam.examples` package.

### 1.3.1 Contents

- *Kollár/Springer C-Section Example*
- *Megson's Closed Section with Branches Example*
- *Kollár/Springer Single Cell Example*
- *Bruhn's Multi-celled Example*
- *Torque-box Example*
- *Hat Stiffener Example*

### 1.3.2 Kollár/Springer C-Section Example

Open sections are the simplest and most common beam type. In this C-Section example from ref.<sup>1</sup>, we calculate, print and plot section properties, create a load case with a vertical shear of 100N and plot its  $N_{xy}$  internal loads. Units for this problem are: m, N and Pa:

```
import abdbeam as ab

sc = ab.Section()
# Create a materials dictionary:
mts = dict()
mts[1] = ab.Laminate()
mts[1].ply_materials[1] = ab.PlyMaterial(0.0001, 148e9, 9.65e9, 4.55e9,
                                          0.3)
mts[1].ply_materials[2] = ab.PlyMaterial(0.0002, 16.39e9, 16.39e9,
                                          38.19e9, 0.801)
mts[1].plies = [[0,2], [0,2], [0,1], [0,1],
                [0,1], [0,1], [0,1], [0,1]]
mts[1].symmetry = 'S'
# Create a points dictionary based on Y and Z point coordinates:
pts = dict()
pts[1] = ab.Point(0.0, 0.0)
pts[2] = ab.Point(0.049, 0.0)
pts[3] = ab.Point(0.049, 0.062)
pts[4] = ab.Point(0.0, 0.062)
# Create a segments dictionary referencing point and material ids:
sgs = dict()
sgs[1] = ab.Segment(1,2,1)
sgs[2] = ab.Segment(2,3,1)
sgs[3] = ab.Segment(3,4,1)
# Point the dictionaries to the section
sc.materials = mts
sc.points = pts
sc.segments = sgs
```

(continues on next page)

---

<sup>1</sup> Kollár LP, Springer GS. Mechanics of composite structures. Cambridge university press; 2003 Feb 17.



(continued from previous page)

```
# Calculate and output section properties
sc.calculate_properties()
sc.summary()
ab.plot_section(sc, figsize=(6.4*0.8, 4.8*0.8))
# Create a single load case and calculate its internal loads
sc.loads[1] = ab.Load(Vz_s=-100)
sc.calculate_internal_loads()
# Plot internal loads
ab.plot_section_loads(sc, 1, int_load_list=['Nxy'],
                      title_list=['Abdbeam - Nxy (N/m)'],
                      figsize=(6.4*0.8, 4.8*0.8))
```

Which prints:

```
Section Summary
=====

Number of points: 4
Number of segments: 3
Number of cells: 0

Centroid
-----
yc      = 3.39937500e-02
zc      = 3.10000000e-02

Shear Center
-----
ys      = 6.92284048e-02
zs      = 3.10000000e-02

Replacement Stiffnesses
-----
EA      = 3.08687460e+07
EIyy    = 2.20044020e+04
EIzz    = 8.18260398e+03
EIyz    = 3.72546093e-12
GJ      = 1.31941376e+01
EImax   = 2.20044020e+04
EImin   = 8.18260398e+03
Angle   = -1.54432286e-14

[P_c] - Beam Stiffness Matrix at the Centroid
-----
[[ 3.08687460e+07  2.87672843e-10 -1.44055909e-10  0.00000000e+00]
 [ 2.87672843e-10  2.20044020e+04  3.72546093e-12  0.00000000e+00]
 [ 1.45837431e-10  5.47177645e-12  8.18260398e+03  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.31941376e+01]]

[W_c] - Beam Compliance Matrix at the Centroid
-----
[[ 3.23952259e-08 -4.23516474e-22  5.70322567e-22  0.00000000e+00]
 [-4.23516474e-22  4.54454522e-05 -2.06908775e-20  0.00000000e+00]
 [-5.77375679e-22 -3.03897581e-20  1.22210485e-04  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  7.57912363e-02]]

[P] - Beam Stiffness Matrix at the Origin
```

(continues on next page)

(continued from previous page)

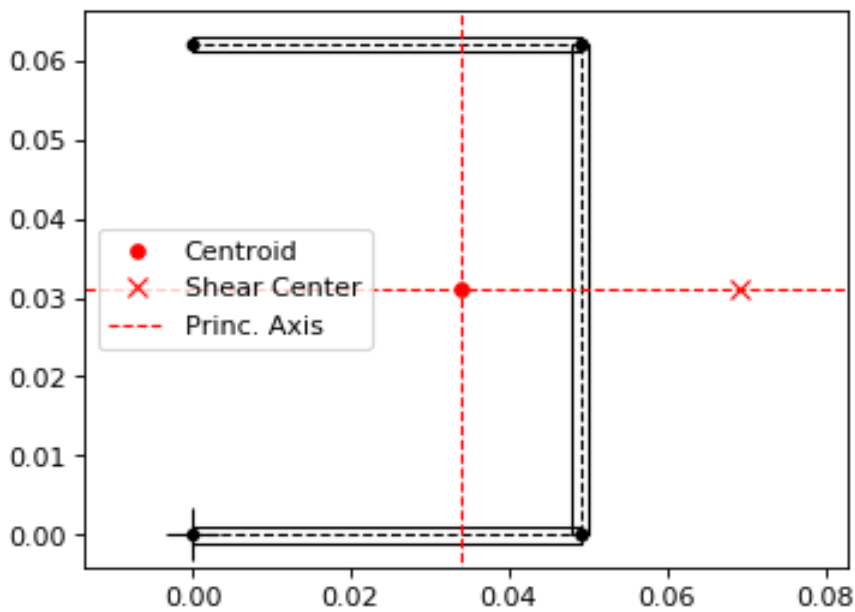
```

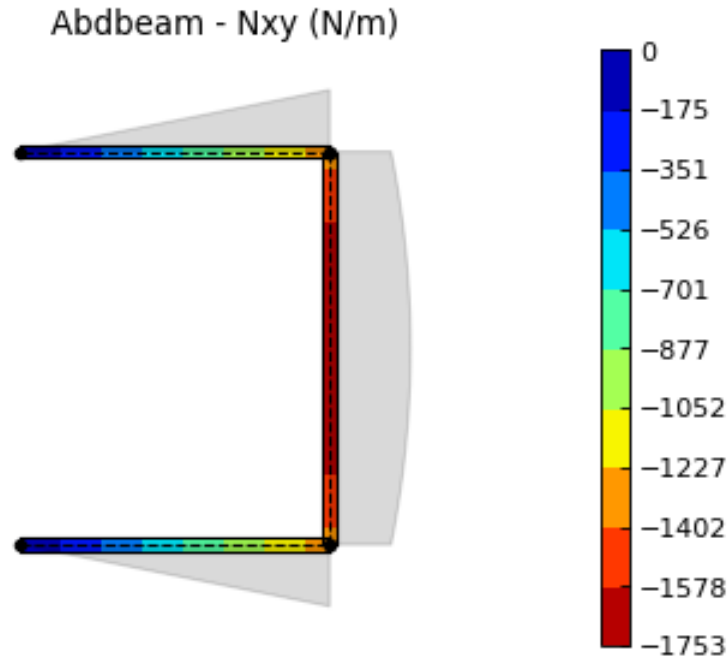
-----
[[3.08687460e+07 9.56931125e+05 1.04934443e+06 0.00000000e+00]
 [9.56931125e+05 5.16692669e+04 3.25296774e+04 0.00000000e+00]
 [1.04934443e+06 3.25296774e+04 4.38537563e+04 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 1.31941376e+01]]

[W] - Beam Compliance Matrix at the Origin
-----
[[ 2.17291691e-07 -1.40880902e-06 -4.15439267e-06 0.00000000e+00]
 [-1.40880902e-06 4.54454522e-05 -2.06908775e-20 0.00000000e+00]
 [-4.15439267e-06 -3.03897581e-20 1.22210485e-04 0.00000000e+00]
 [ 0.00000000e+00 0.00000000e+00 0.00000000e+00 7.57912363e-02]]

```

And plots:





Back to [Contents](#).

### 1.3.3 Megson's Closed Section with Branches Example

Megson's book (ref<sup>2</sup>) example 19.1 is dedicated to calculating the shear flow distribution for a beam combining open and closed section elements. Units for this problem are: mm, N and MPa.

We start creating the section and calculating its properties:

```
import abdbeam as ab

sc = ab.Section()
# Create a dictionary for the isotropic material
mts = dict()
mts[1] = ab.Isotropic(2, 70000, 0.3)
# Create a points dictionary based on Y and Z point coordinates
pts = dict()
pts[1] = ab.Point(0, -200)
pts[2] = ab.Point(-100, -200)
pts[3] = ab.Point(-100, 0)
pts[4] = ab.Point(-200, 0)
pts[5] = ab.Point(-200, -100)
pts[6] = ab.Point(0, 0)
pts[7] = ab.Point(100, -200)
pts[8] = ab.Point(100, 0)
pts[9] = ab.Point(200, 0)
pts[10] = ab.Point(200, -100)
# Create a segments dictionary referencing point and material ids
sgs = dict()
sgs[1] = ab.Segment(1, 2, 1)
```

(continues on next page)

<sup>2</sup> Megson TH. Aircraft structures for engineering students. Butterworth-Heinemann; 2016 Oct 17.

(continued from previous page)

```

sgs[2] = ab.Segment(2,3,1)
sgs[3] = ab.Segment(3,4,1)
sgs[4] = ab.Segment(4,5,1)
sgs[5] = ab.Segment(3,6,1)
sgs[6] = ab.Segment(1,7,1)
sgs[7] = ab.Segment(7,8,1)
sgs[8] = ab.Segment(8,9,1)
sgs[9] = ab.Segment(9,10,1)
sgs[10] = ab.Segment(8,6,1)
# Point the dictionaries to the section
sc.materials = mts
sc.points = pts
sc.segments = sgs
# Calculate section properties
sc.calculate_properties()

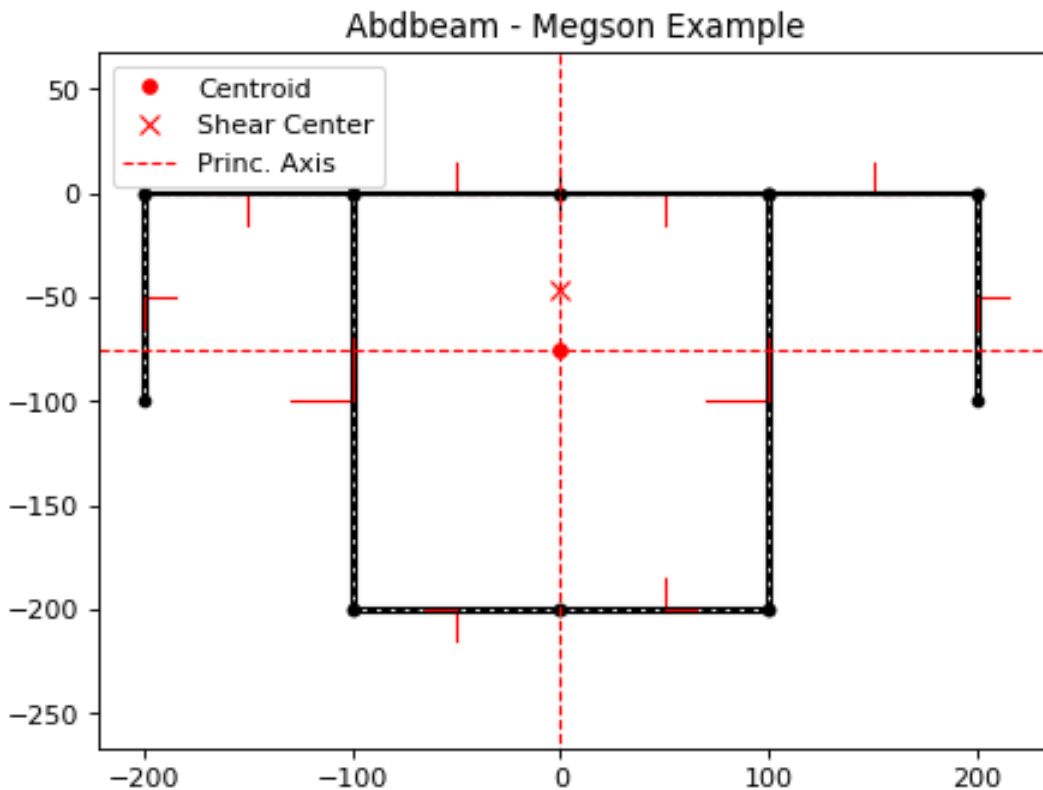
```

Next, we are going to plot the section showing its segments' orientations, as they are essential to understand shear signs:

```

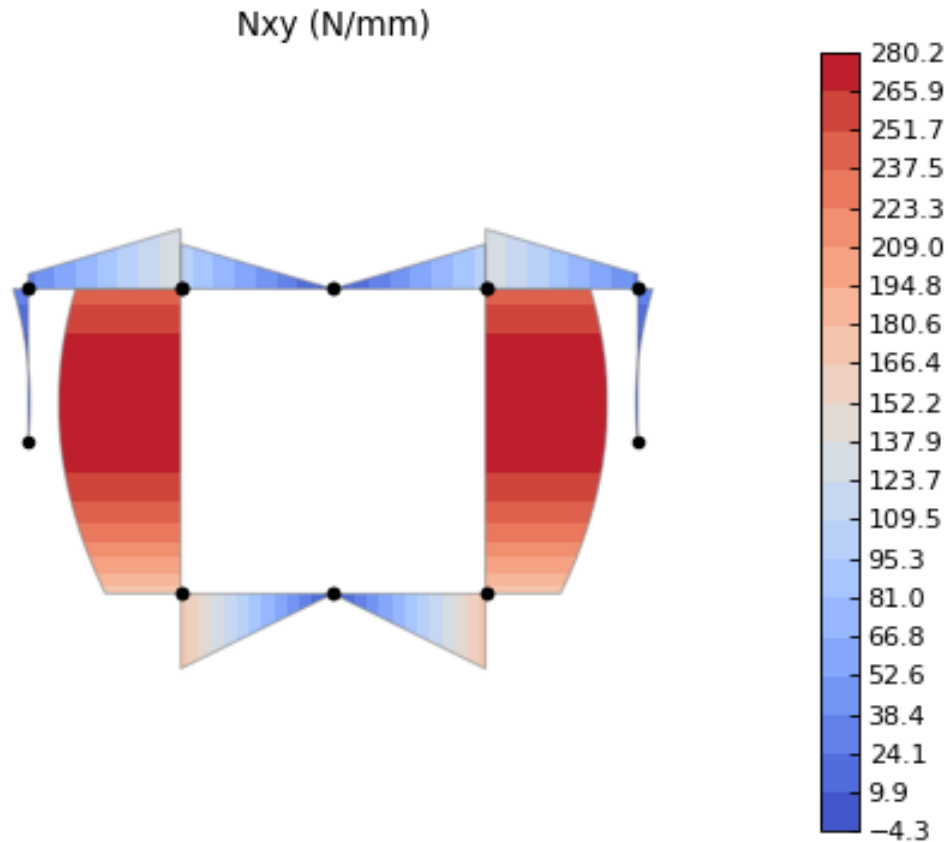
# Plot the section
ab.plot_section(sc, segment_coord=True, title='Abdbeam - Megson Example')

```



*Abdbeam* by default plots positive diagrams towards the top side of the laminate and negative diagrams towards the bottom side. To change individual diagram directions, the parameter `diagram_factor_list` can be used with factors 1.0 or -1.0 as desired. This is also a rather large cross section with thin laminates, so plotting a contour inside the thickness will be hardly visible. To provide a clearer internal load view, we'll use a contour diagram:

```
#Create load case and calculate its internal loads:
sc.loads[1] = ab.Load(Vz_s=100000)
sc.calculate_internal_loads()
ab.plot_section_loads(sc, 1, segment_contour=False, diagram=True,
                     diagram_contour=True, diagram_alpha=1.0,
                     contour_levels=20, contour_color='coolwarm',
                     diagram_factor_list=[1,1,-1,-1,1,-1,-1,1,1,-1],
                     thickness=False, int_load_list=['Nxy'],
                     title_list=['Nxy (N/mm)'])
```



Back to [Contents](#).

### 1.3.4 Kollár/Springer Single Cell Example

In this single rectangular cell example from ref.<sup>1</sup>, the laminate is asymmetric. This requires special attention when defining the point connectivity of each segment, so that the bottom and top plies are at their intended sides. We'll calculate, print and plot section properties, create a load case with combined external loads and plot its  $N_x$  and  $N_{xy}$  internal loads in a single plot. Units for this problem are: m, N and Pa:

```
import abdbeam as ab

sc = ab.Section()
# Create a materials dictionary:
mts = dict()
```

(continues on next page)

(continued from previous page)

```

mts[1] = ab.Laminate()
mts[1].ply_materials[1] = ab.PlyMaterial(0.0001, 148e9, 9.65e9, 4.55e9,
                                          0.3)
mts[1].ply_materials[2] = ab.PlyMaterial(0.0002, 16.39e9, 16.39e9,
                                          38.19e9, 0.801)

mts[1].plies = [[0,1]]*10 + [[45,1]]*10
mts[1].symmetry = 'T'
# Create a points dictionary based on Y and Z point coordinates:
pts = dict()
pts[1] = ab.Point(-0.025, -0.035)
pts[2] = ab.Point(0.025, -0.035)
pts[3] = ab.Point(0.025, 0.035)
pts[4] = ab.Point(-0.025, 0.035)
# Create a segments dictionary referencing point and material ids:
sgs = dict()
sgs[1] = ab.Segment(1,2,1)
sgs[2] = ab.Segment(2,3,1)
sgs[3] = ab.Segment(3,4,1)
sgs[4] = ab.Segment(4,1,1)
# Point the dictionaries to the section
sc.materials = mts
sc.points = pts
sc.segments = sgs
# Calculate and output section properties
sc.calculate_properties()
sc.summary()
ab.plot_section(sc, segment_coord=True, figsize=(6.4*0.8, 4.8*0.8))
# Create a single load case and calculate its internal loads
sc.loads[1] = ab.Load(Px=200, Mz=10, Vz_s=-100)
sc.calculate_internal_loads()
# Plot internal loads
ab.plot_section_loads(sc, 1, int_load_list=['Nx', 'Nxy'],
                     title_list=['Abdbeam - Nx (N/m)',
                                'Abdbeam - Nxy (N/m)'], figsize=(6.4*0.8, 4.8*0.8))

```

Which prints:

```

Section Summary
=====

Number of points: 4
Number of segments: 4
Number of cells: 1

Centroid
-----
yc      = 0.00000000e+00
zc      = 0.00000000e+00

Shear Center
-----
ys      = 0.00000000e+00
zs      = -6.93889390e-19

Replacement Stiffnesses
-----
EA      = 3.99342118e+07

```

(continues on next page)

(continued from previous page)

```

EIyy  = 2.94941190e+04
EIzz  = 1.79757984e+04
EIyz  = 0.00000000e+00
GJ     = 4.10520186e+03
EImax = 2.94941190e+04
EImin = 1.79757984e+04
Angle  = -0.00000000e+00

```

[P\_c] - Beam Stiffness Matrix at the Centroid

```

-----
[[ 3.99342118e+07 -0.00000000e+00 -0.00000000e+00  6.75301020e+04]
 [ 0.00000000e+00  2.94941190e+04  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.79757984e+04  0.00000000e+00]
 [ 6.75301020e+04  0.00000000e+00  0.00000000e+00  4.10520186e+03]]

```

[W\_c] - Beam Compliance Matrix at the Centroid

```

-----
[[ 2.57576953e-08  0.00000000e+00  0.00000000e+00 -4.23711147e-07]
 [ 0.00000000e+00  3.39050642e-05  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  5.56303524e-05  0.00000000e+00]
 [-4.23711147e-07  0.00000000e+00  0.00000000e+00  2.50563381e-04]]

```

[P] - Beam Stiffness Matrix at the Origin

```

-----
[[ 3.99342118e+07 -0.00000000e+00 -0.00000000e+00  6.75301020e+04]
 [ 0.00000000e+00  2.94941190e+04  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.79757984e+04  0.00000000e+00]
 [ 6.75301020e+04  0.00000000e+00  0.00000000e+00  4.10520186e+03]]

```

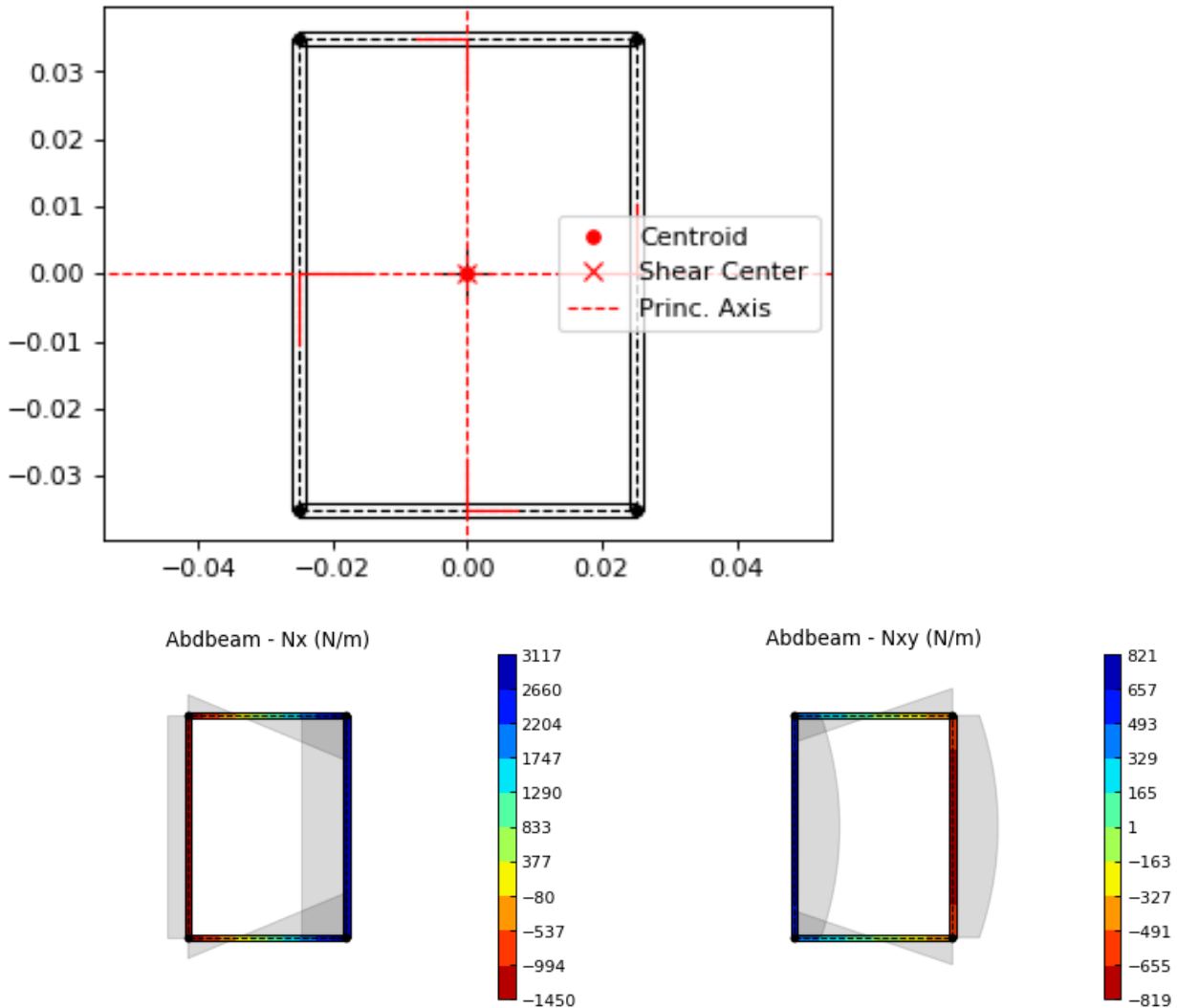
[W] - Beam Compliance Matrix at the Origin

```

-----
[[ 2.57576953e-08  0.00000000e+00  0.00000000e+00 -4.23711147e-07]
 [ 0.00000000e+00  3.39050642e-05  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  5.56303524e-05  0.00000000e+00]
 [-4.23711147e-07  0.00000000e+00  0.00000000e+00  2.50563381e-04]]

```

And plots:



[Back to Contents.](#)

### 1.3.5 Bruhn's Multi-celled Example

Most idealized solutions to aerospace beam problems assume that booms are connected to segments that can only resist shear. Shear connectors can be used to achieve this, since they only use as inputs a material shear modulus and its thickness.

Consider the cross section with 5 cells from Bruhn's example A15.12 (ref<sup>3</sup>). Two important assumptions are made: the cells segments (beam walls) can only resist shear and the points (beam flanges) develop all the bending resistance. As stated above, in *Abdbeam* we can enforce these assumptions by modeling the segments with shear connector materials and the section points with EA and/or GJ properties (booms). A  $V_z$  load of 1000lbf is applied to the section and only the shear flows (Nxy) at the segments and shear center are of interest. Units for this problem are: in, lbf and psi.

We'll start creating the section, calculating its properties and plotting its section. Notice that booms are represented with additional circles around points:

<sup>3</sup> Bruhn EF, Bollard RJ. Analysis and design of flight vehicle structures. Indianapolis: Jacobs Publishing; 1973 Jun.

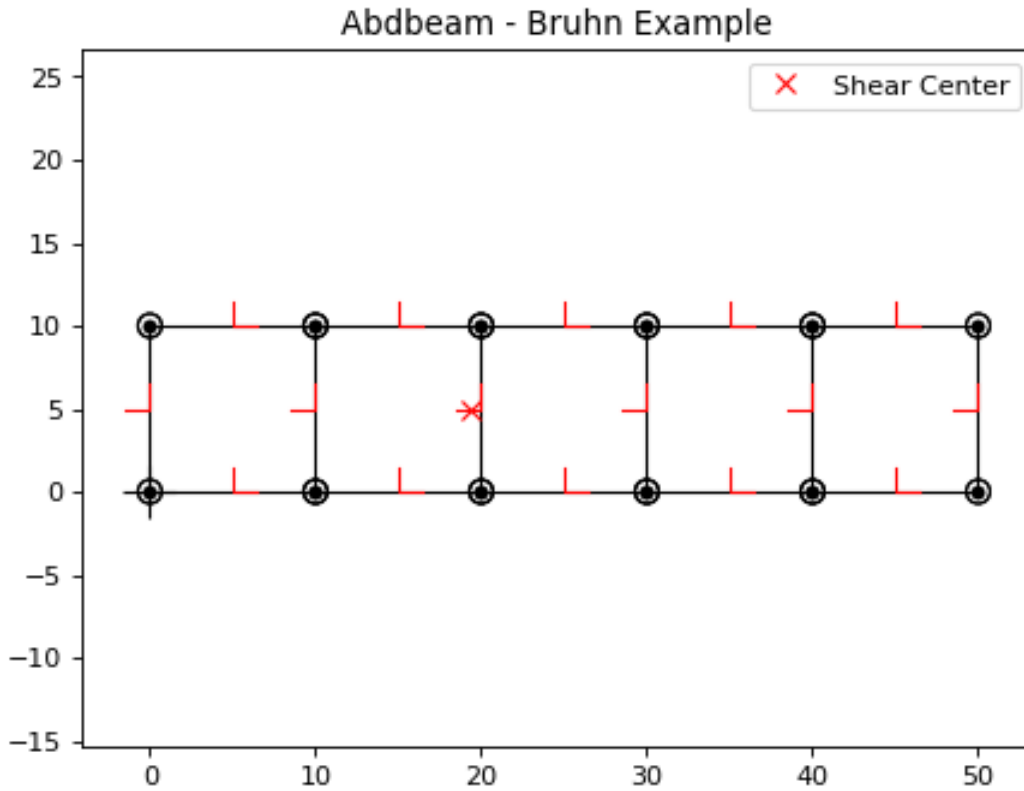


```

import abdbeam as ab

sc = ab.Section()
# Create a dictionary for the shear connector materials
mts = dict()
mts[1] = ab.ShearConnector(0.03, 3846154)
mts[2] = ab.ShearConnector(0.04, 3846154)
mts[3] = ab.ShearConnector(0.05, 3846154)
mts[4] = ab.ShearConnector(0.064, 3846154)
# Create a points dictionary based on Y and Z point coordinates
pts = dict()
pts[1] = ab.Point(0,0,2e7,0,'a')
pts[2] = ab.Point(10,0,1e7,0,'b')
pts[3] = ab.Point(20,0,5e6,0,'c')
pts[4] = ab.Point(30,0,5e6,0,'d')
pts[5] = ab.Point(40,0,5e6,0,'e')
pts[6] = ab.Point(50,0,1e7,0,'f')
pts[11] = ab.Point(0,10,2e7,0,'a_')
pts[12] = ab.Point(10,10,1e7,0,'b_')
pts[13] = ab.Point(20,10,5e6,0,'c_')
pts[14] = ab.Point(30,10,5e6,0,'d_')
pts[15] = ab.Point(40,10,5e6,0,'e_')
pts[16] = ab.Point(50,10,1e7,0,'f_')
# Create a segments dictionary referencing point and material ids
sgs = dict()
sgs[1] = ab.Segment(1,2,2,'Bottom 1')
sgs[2] = ab.Segment(2,3,2,'Bottom 2')
sgs[3] = ab.Segment(3,4,2,'Bottom 3')
sgs[4] = ab.Segment(4,5,1,'Bottom 4')
sgs[5] = ab.Segment(5,6,1,'Bottom 5')
sgs[11] = ab.Segment(11,12,2,'Top 1')
sgs[12] = ab.Segment(12,13,2,'Top 2')
sgs[13] = ab.Segment(13,14,2,'Top 3')
sgs[14] = ab.Segment(14,15,1,'Top 4')
sgs[15] = ab.Segment(15,16,1,'Top 5')
sgs[21] = ab.Segment(1,11,4,'Web 1')
sgs[22] = ab.Segment(2,12,3,'Web 2')
sgs[23] = ab.Segment(3,13,2,'Web 3')
sgs[24] = ab.Segment(4,14,2,'Web 4')
sgs[25] = ab.Segment(5,15,1,'Web 5')
sgs[26] = ab.Segment(6,16,1,'Web 6')
# Point the dictionaries to the section
sc.materials = mts
sc.points = pts
sc.segments = sgs
# Calculate section properties
sc.calculate_properties()
# Plot the section
ab.plot_section(sc, centroid=False, princ_dir=False, thickness=False,
               segment_coord=True, title='Abdbeam - Bruhn Example')

```



Next we'll create the load case, calculate its internal loads and obtain the shear flows accessing the internal loads Pandas dataframe `sc.sgs_int_lds_df` directly. Since the segments develop no bending resistance, the shear flow between adjacent points will be constant, and the average  $N_{xy}$  per segment is appropriate:

```
#Create load case and calculate its internal loads:
sc.loads[1] = ab.Load(Vz_s=1000)
sc.calculate_internal_loads()
# Print the shear flows Nxy for all segments
df = sc.sgs_int_lds_df
print(df[['Segment_Id', ''], ('Nxy', 'Avg')])
# Print the shear center location
print('')
print('Shear center is at y = {:.8e}'.format(sc.ys))
```

Which prints:

	Segment_Id	Nxy Avg
0	1	2.878645
1	2	2.096924
2	3	0.210851
3	4	-1.253519
4	5	-2.586107
5	11	-2.878645
6	12	-2.096924
7	13	-0.210851
8	14	1.253519
9	15	2.586107
10	21	33.484992

(continues on next page)

(continued from previous page)

11	22	18.963538
12	23	10.976982
13	24	10.555279
14	25	10.423497
15	26	15.595711

Shear center is at  $y = 1.93602680e+01$

Back to [Contents](#).

### 1.3.6 Torque-box Example

Shear connectors can also be used to model mechanical or bonded joints. This allows the representation of multiple elements of a cross section by their mid-surfaces, usually at the price of creating additional small enclosed areas (cells) between pairs of adjacent shear connector segments.

In this example, we create a torque-box using this modeling approach, which could be extended to complex cross sections such as those found in wings, vertical stabilizers, horizontal stabilizers and others (to use aerospace examples). Here we have a C-Section forward spar and a I-Section rear spar with rows of fasteners (shear connectors) connecting them to top and bottom skins. Notice that this modeling approach added two additional closed cells near the rear spar (see the [Hat Stiffener Example](#) for a more detailed discussion on the effects of such small closed cells in the section torsional stiffness). Units for this problem are: in, lbf and psi.

First we create the cross section, calculate its properties and plot it:

```
import abdbeam as ab

sc = ab.Section()
# Create a dictionary to store ply materials shared by laminates
ply_mts = dict()
ply_mts[1] = ab.PlyMaterial(0.0075, 1.149e7, 1.149e7, 6.6e5, 0.04)
# Create the materials dictionary for the laminates and shear connector:
mts = dict()
mts[1] = ab.Laminate()
mts[1].ply_materials[1] = ply_mts[1]
mts[1].plies = [[45,1], [-45,1]]*2 + [[0,1]]*3
mts[1].symmetry = 'S'
mts[2] = ab.ShearConnector(0.25, 6381760)
# Create a points dictionary based on Y and Z point coordinates:
pts = dict()
pts[1] = ab.Point(1,-4)
pts[2] = ab.Point(2,-4)
pts[3] = ab.Point(12,-4)
pts[4] = ab.Point(14,-4)
pts[5] = ab.Point(15,-4)
pts[11] = ab.Point(1,1.21)
pts[12] = ab.Point(2,1.21)
pts[13] = ab.Point(12,1.21)
pts[14] = ab.Point(14,1.21)
pts[15] = ab.Point(15,1.21)
pts[21] = ab.Point(1,-3.895)
pts[22] = ab.Point(2,-3.895)
pts[23] = ab.Point(3,-3.895)
pts[24] = ab.Point(3,1.105)
pts[25] = ab.Point(2,1.105)
```

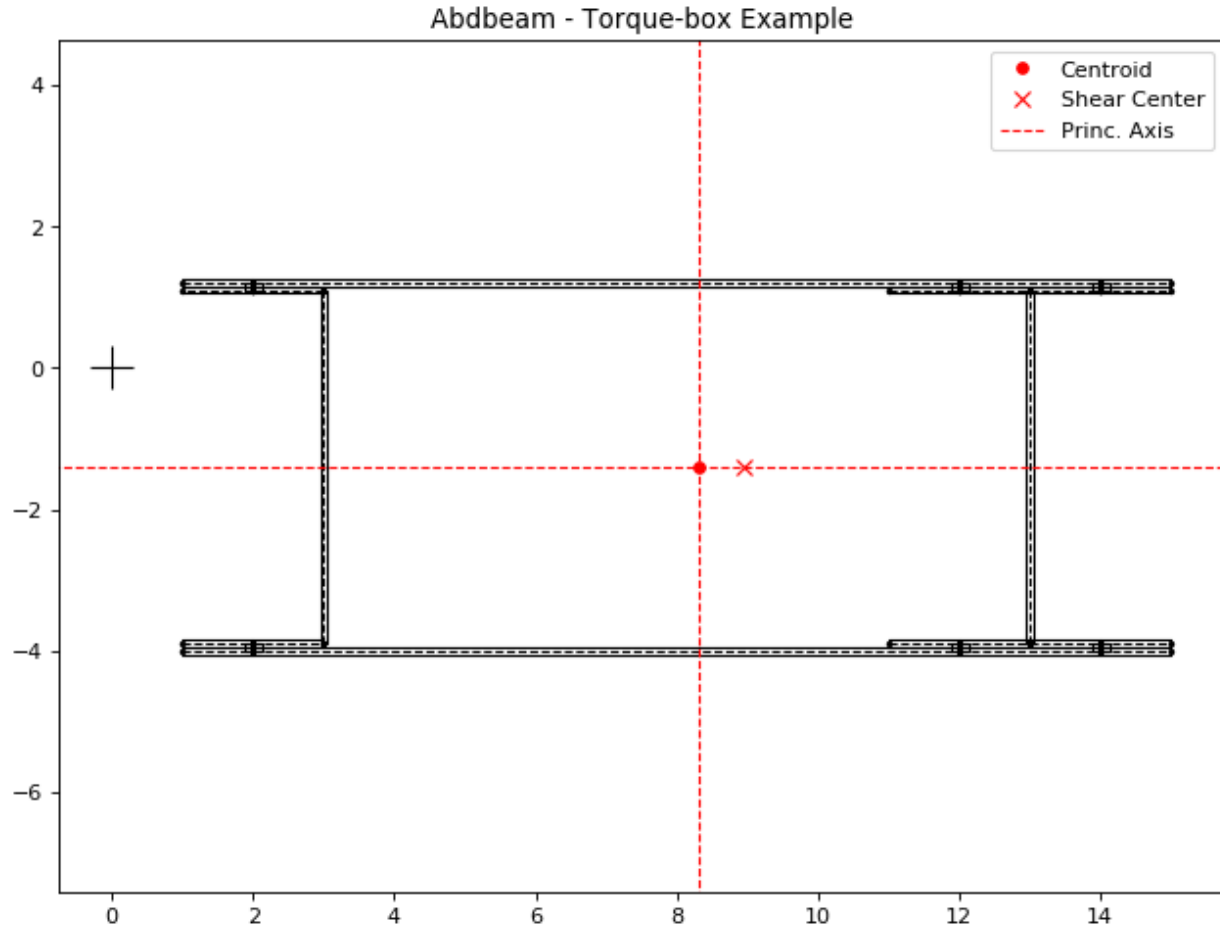
(continues on next page)

(continued from previous page)

```

pts[26] = ab.Point(1,1.105)
pts[31] = ab.Point(11,-3.895)
pts[32] = ab.Point(12,-3.895)
pts[33] = ab.Point(13,-3.895)
pts[34] = ab.Point(14,-3.895)
pts[35] = ab.Point(15,-3.895)
pts[36] = ab.Point(11,1.105)
pts[37] = ab.Point(12,1.105)
pts[38] = ab.Point(13,1.105)
pts[39] = ab.Point(14,1.105)
pts[40] = ab.Point(15,1.105)
# Create a segments dictionary referencing point and material ids:
sgs = dict()
sgs[1] = ab.Segment(1,2,1,'Bottom Skin')
sgs[2] = ab.Segment(2,3,1,'Bottom Skin')
sgs[3] = ab.Segment(3,4,1,'Bottom Skin')
sgs[4] = ab.Segment(4,5,1,'Bottom Skin')
sgs[11] = ab.Segment(11,12,1,'Top Skin')
sgs[12] = ab.Segment(12,13,1,'Top Skin')
sgs[13] = ab.Segment(13,14,1,'Top Skin')
sgs[14] = ab.Segment(14,15,1,'Top Skin')
sgs[21] = ab.Segment(21,22,1,'Fwd Spar')
sgs[22] = ab.Segment(22,23,1,'Fwd Spar')
sgs[23] = ab.Segment(23,24,1,'Fwd Spar')
sgs[24] = ab.Segment(24,25,1,'Fwd Spar')
sgs[25] = ab.Segment(25,26,1,'Fwd Spar')
sgs[31] = ab.Segment(31,32,1,'Rear Spar')
sgs[32] = ab.Segment(32,33,1,'Rear Spar')
sgs[33] = ab.Segment(33,34,1,'Rear Spar')
sgs[34] = ab.Segment(34,35,1,'Rear Spar')
sgs[35] = ab.Segment(33,38,1,'Rear Spar')
sgs[36] = ab.Segment(36,37,1,'Rear Spar')
sgs[37] = ab.Segment(37,38,1,'Rear Spar')
sgs[38] = ab.Segment(38,39,1,'Rear Spar')
sgs[39] = ab.Segment(39,40,1,'Rear Spar')
sgs[91] = ab.Segment(2,22,2,'Connector')
sgs[92] = ab.Segment(3,32,2,'Connector')
sgs[93] = ab.Segment(4,34,2,'Connector')
sgs[94] = ab.Segment(25,12,2,'Connector')
sgs[95] = ab.Segment(37,13,2,'Connector')
sgs[96] = ab.Segment(39,14,2,'Connector')
# Point the dictionaries to the section
sc.materials = mts
sc.points = pts
sc.segments = sgs
# Calculate section properties
sc.calculate_properties()
# Plot the section
ab.plot_section(sc, pt_size=2, title='Abdbeam - Torque-box Example',
               figsize=(6.4*1.5, 4.8*1.5))

```



Next we're going to create 7 load cases with axial, bending, torque and shear loads integrated at the section origin.  $Y_p$ ,  $z_p$ ,  $y_s$  and  $z_s$  are then all equal to the default zero and for this reason don't need to be explicitly entered. Notice that if  $Px_c$  were used instead of  $Px$ , the axial load would always be assumed to act on the centroid. By using a  $Px$  though, a moment arm results from the load application point distance to the calculated centroid. Similarly, if  $Vy_s$  and  $Vz_s$  were used instead of  $Vy$  and  $Vz$ , the shear loads would be assumed to always act at the shear center, creating no additional torque in the section. The approach chosen for this example is the typical case for a torque box: loads are integrated at a defined point in space and sizing proceeds changing centroid and shear center locations.

Create the load cases and calculate their internal loads:

```
# Create load cases and calculate their internal loads
sc.loads[8] = ab.Load(Px=17085, My=-140914, Mz=-7208, Tx=1595, Vy=4727, Vz=-1661)
sc.loads[4] = ab.Load(Px=11854, My=-89211, Mz=-33716, Tx=-57488, Vy=5684, Vz=394)
sc.loads[1] = ab.Load(Px=2395, My=-83206, Mz=210099, Tx=-43162, Vy=1316, Vz=407)
sc.loads[10] = ab.Load(Px=-7458, My=-15571, Mz=-96370, Tx=-3615, Vy=564, Vz=-369)
sc.loads[2] = ab.Load(Px=1000, My=-30865, Mz=180498, Tx=11653, Vy=-7001, Vz=-189)
sc.loads[3] = ab.Load(Px=-281, My=133314, Mz=-123966, Tx=324, Vy=9389, Vz=-1514)
sc.loads[6] = ab.Load(Px=299, My=40658, Mz=101677, Tx=7102, Vy=9214, Vz=-3545)
sc.calculate_internal_loads()
```

Now let's say we are analyzing the rear spar and would like to find the critical compressive  $N_x$  load among all cases, identify its load case id and segment. Since the calculated internal loads are stored in the Pandas dataframe `sc.sgs_int_lds_df`, we can use Pandas methods to achieve this:

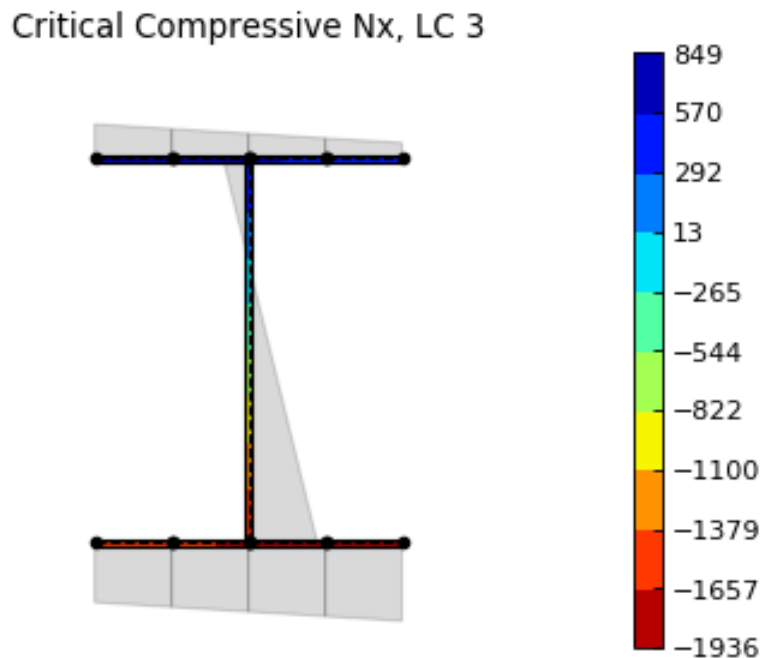
```
# Use Pandas methods to get info on the critical spar compressive Nx
df = sc.sgs_int_lds_df
spar_sgs = range(31,40)
df = df[df['Segment_Id'].isin(spar_sgs)]
idx = df[('Nx', 'Min')].idxmin()
min_Nx = round(df.loc[idx, ('Nx', 'Min')],1)
min_sg = int(df.loc[idx, 'Segment_Id'])
min_lc = int(df.loc[idx, 'Load_Id'])
print(('Minimum rear spar Nx is {}, from segment {}, load case {}'.format(min_Nx, min_sg, min_lc))
```

Which prints:

```
Minimum rear spar Nx is -1935.9, from segment 34, load case 3
```

We can finish the example plotting the Nx from this critical load case. To do this, we're going to filter the rear spar from the rest of the cross section using the parameter `plot_sgs` and get the load case id from the variable `min_lc` defined on the previous code block:

```
# Plot the critical compressive case Nx internal loads
ab.plot_section_loads(sc, min_lc, int_load_list=['Nx'],
                    title_list=['Critical Compressive Nx, LC '+
                                str(min_lc)], plot_sgs=range(31,40),
                    figsize=(6.4*0.8, 4.8*0.8))
```



Back to [Contents](#).

### 1.3.7 Hat Stiffener Example

In this example we create a hat stiffener and a skin, both represented at their laminates' mid-planes. *Abdbeam* sections cannot have floating segments and we want to capture the hat's closed cell contribution to the section's GJ, so we chose

here to connect the hat plies to the bottom skin using `ShearConnector` materials. For these connector's properties, we entered a `t` and `G` that matches compliance term  $\alpha_{66}$  ( $=1/(G*t)$ ) of the skin laminate material. Notice that, by also connecting the left and right outermost cap points, we added two other cells to the analysis. The areas of these two cells are small and they only increase the total section `GJ` by 3% (compared to a section removing connectors 91 and 94). Connecting segments that are co-cured or bonded, on the other hand, tend to better represent the  $N_{xy}$  shear distribution in complex sections. This example also shows multiple ways to define a stacking sequence using python lists capabilities. Units for this problem are: in, lbf and psi.

We'll start creating the section, calculating its properties and showing a summary of these properties:

```
import abdbeam as ab

sc = ab.Section()
# Create a dictionary to store ply materials shared by laminates
ply_mts = dict()
ply_mts[1] = ab.PlyMaterial(0.0075, 2.147e7, 1.4e6, 6.6e5, 0.3)
ply_mts[2] = ab.PlyMaterial(0.0075, 1.149e7, 1.149e7, 6.6e5, 0.04)
# Create the materials dictionary for the laminates and shear connector:
mts = dict()
mts[1] = ab.Laminate()
mts[1].ply_materials[2] = ply_mts[2]
mts[1].plies = [[45,2], [-45,2]] + [[0,2]]*3
mts[1].symmetry = 'S'
mts[2] = ab.Laminate()
mts[2].ply_materials[2] = ply_mts[2]
mts[2].plies = [[45,2], [-45,2]]*2 + [[0,2]]
mts[2].symmetry = 'S'
mts[3] = ab.Laminate()
mts[3].ply_materials[1] = ply_mts[1]
mts[3].ply_materials[2] = ply_mts[2]
mts[3].plies = [[45,2], [-45,2]] + [[0,1]]*3 + [[0,2]] + [[0,1]]*2
mts[3].symmetry = 'SM'
mts[4] = ab.ShearConnector(0.075, 2605615)
# Create a points dictionary based on Y and Z point coordinates:
pts = dict()
pts[1] = ab.Point(-2, 0)
pts[2] = ab.Point(-1, 0)
pts[3] = ab.Point(1, 0)
pts[4] = ab.Point(2, 0)
pts[5] = ab.Point(-2, 0.075)
pts[6] = ab.Point(-1, 0.075)
pts[7] = ab.Point(-0.35, 0.8)
pts[8] = ab.Point(0.35, 0.8)
pts[9] = ab.Point(1, 0.075)
pts[10] = ab.Point(2, 0.075)
# Create a segments dictionary referencing point and material ids:
sgs = dict()
sgs[1] = ab.Segment(1,2,1,'Skin_Left')
sgs[2] = ab.Segment(2,3,1,'Skin_Center')
sgs[3] = ab.Segment(3,4,1,'Skin_Right')
sgs[10] = ab.Segment(5,6,2,'Hat_Left_Foot')
sgs[11] = ab.Segment(6,7,2,'Hat_Left_Web')
sgs[12] = ab.Segment(7,8,3,'Hat_Top')
sgs[13] = ab.Segment(8,9,2,'Hat_Right_Web')
sgs[14] = ab.Segment(9,10,2,'Hat_Right_Foot')
sgs[91] = ab.Segment(1,5,4,'Connector_1')
sgs[92] = ab.Segment(2,6,4,'Connector_1')
sgs[93] = ab.Segment(3,9,4,'Connector_1')
```

(continues on next page)

(continued from previous page)

```

sgs[94] = ab.Segment(4,10,4,'Connector_1')
# Point the dictionaries to the section
sc.materials = mts
sc.points = pts
sc.segments = sgs
# Calculate section properties
sc.calculate_properties()
sc.summary()

```

Which prints:

```

Section Summary
=====

Number of points: 10
Number of segments: 12
Number of cells: 3

Centroid
-----
yc      = 0.00000000e+00
zc      = 2.51301984e-01

Shear Center
-----
ys      = 1.83880688e-16
zs      = 4.13936823e-01

Replacement Stiffnesses
-----
EA       = 5.43010577e+06
EIyy     = 6.22690978e+05
EIzz     = 5.79683101e+06
EIyz     = 7.27595761e-12
GJ       = 2.71545365e+05
EImax    = 5.79683101e+06
EImin    = 6.22690978e+05
Angle    = 8.05702321e-17

[P_c] - Beam Stiffness Matrix at the Centroid
-----
[[ 5.43010577e+06 -1.79003360e-10 -2.09160066e-27  0.00000000e+00]
 [ 0.00000000e+00  6.22690978e+05  7.27595761e-12  0.00000000e+00]
 [ 0.00000000e+00 -1.45519152e-11  5.79683101e+06  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  2.71545365e+05]]

[W_c] - Beam Compliance Matrix at the Centroid
-----
[[ 1.84158476e-07  5.29395592e-23  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  1.60593302e-06 -2.01570488e-24  0.00000000e+00]
 [ 0.00000000e+00  4.03140976e-24  1.72508048e-07  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  3.68262592e-06]]

[P] - Beam Stiffness Matrix at the Origin
-----
[[ 5.43010577e+06  1.36459635e+06 -3.21443370e-27 -0.00000000e+00]
 [ 1.36459635e+06  9.65616750e+05  7.27595761e-12  0.00000000e+00]

```

(continues on next page)



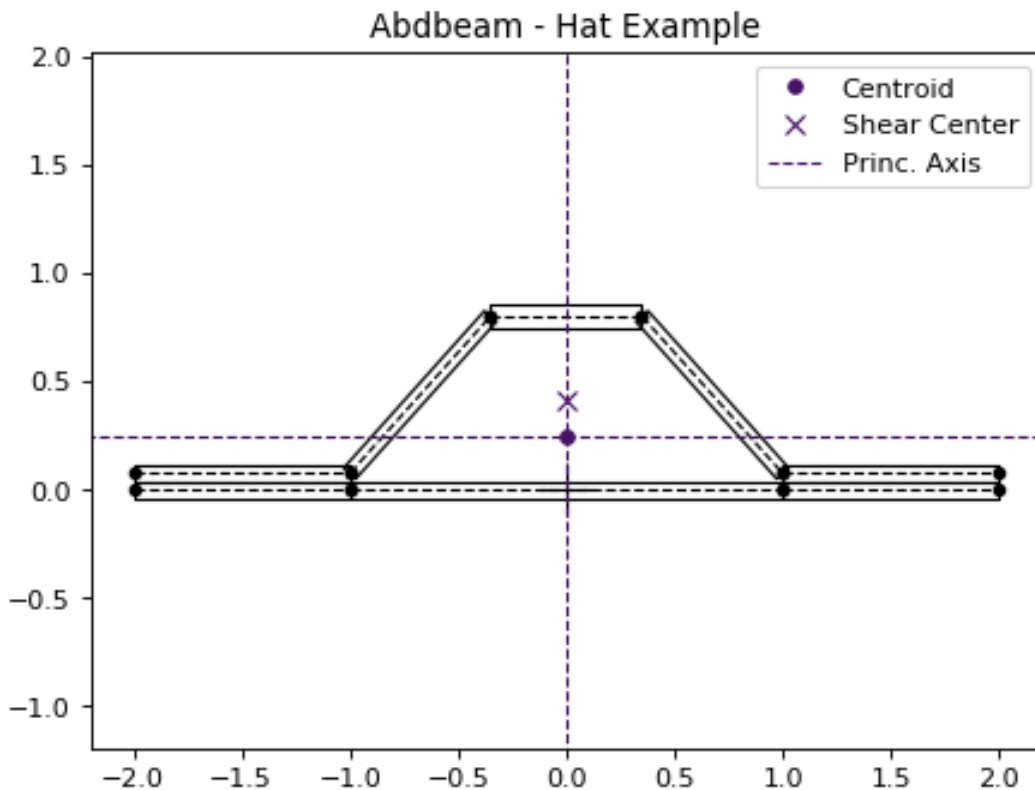
(continued from previous page)

```
[ 1.36115257e-27 -1.45519152e-11  5.79683101e+06  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  2.71545365e+05]]

[W] - Beam Compliance Matrix at the Origin
-----
[[ 2.85577461e-07 -4.03574153e-07  5.06550636e-25  0.00000000e+00]
 [-4.03574153e-07  1.60593302e-06 -2.01570488e-24  0.00000000e+00]
 [-1.01310127e-24  4.03140976e-24  1.72508048e-07  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  3.68262592e-06]]
```

Next, we'll plot the section hiding segments 91 to 94 (the connectors), since we don't care about their visual representation:

```
# Plot the section
ab.plot_section(sc, filter_sgs=[91,92,93,94],
               title='Abdbeam - Hat Example',
               prop_color='#471365')
```



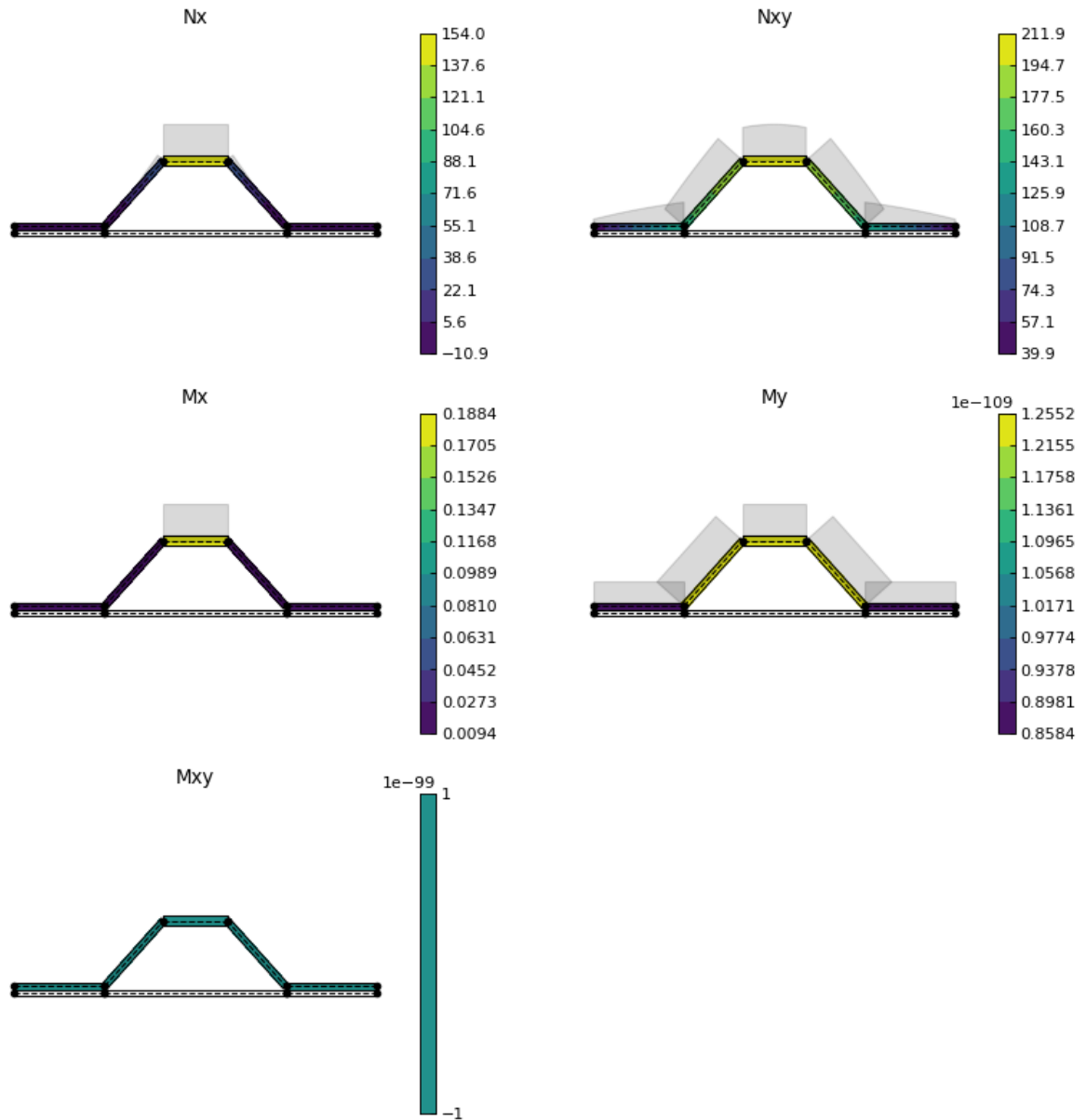
Next, we'll create two load cases and plot the first case (a rather unusual load case for a stiffener, for illustration purposes only). Notice that in this plot we listed only the cap segments to show results and continued to filter the connectors:

```
# Create load cases and calculate their internal loads
sc.loads[1] = ab.Load(My=100, Vy_s=1000)
sc.loads[2] = ab.Load(Tx=100)
sc.calculate_internal_loads()
# Plot internal loads
ab.plot_section_loads(sc, 1, contour_color = 'viridis',
```

(continues on next page)

(continued from previous page)

```
result_sgs=[10,11,12,13,14],
figsize=(6.4*0.8, 4.8*0.8),
diagram_scale=0.5, filter_sgs=[91,92,93,94])
```



Back to [Contents](#).

## References

## 1.4 Theory

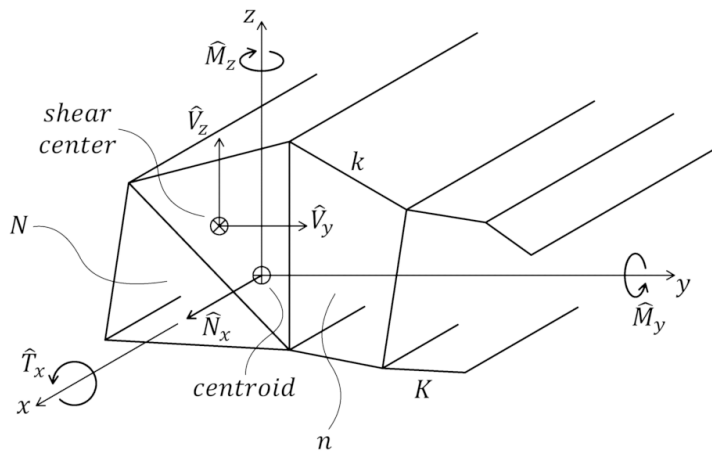
*Abdbeam* uses a thin-walled anisotropic composite beam theory that includes closed cells with open branches and booms. For the detailed theory behind *Abdbeam*, Ref.<sup>1</sup> is the most complete reference. Ref.<sup>2</sup> and Ref.<sup>3</sup> are also great references on its originating theory.

### 1.4.1 Hypothesis and Limitations

- Prismatic thin-walled beams undergoing small deformations;
- Materials behave in a linearly elastic manner;
- Bernoulli-Navier hypothesis: originally plane cross sections of a beam undergoing bending remain plane and perpendicular to the axis of the beam;
- The effects of shear deformation and restrained warping are neglected.

### 1.4.2 Sign Conventions

The figure below show the sign conventions for the cross section axis, its normal force, bending moments, torque and transverse shear forces:



**Warning:** The  $M_z$  sign convention used in *Abdbeam* requires special attention. As seen above, the sign of the bending moments is that they are positive when they induce tension in the positive  $yz$  quadrant of the beam cross section (as seen in Ref.<sup>1</sup> to Ref.<sup>4</sup>). In contrast, Finite Element Analysis software packages commonly adopt a right-hand rule to define the tension and compression signs of a positive  $M_z$  bending moment. Remember to multiply by -1 cross sectional  $M_z$  loads obtained from these sources.

<sup>1</sup> Victorazzo DS, De Jesus A. A Kollár and Pluzsik anisotropic composite beam theory for arbitrary multicelled cross sections. *Journal of Reinforced Plastics and Composites*. 2016 Dec;35(23):1696-711.

<sup>2</sup> Kollár LP, Springer GS. *Mechanics of composite structures*. Cambridge university press; 2003 Feb 17.

<sup>3</sup> Kollár LP and Pluzsik A. Analysis of thin-walled composite beams with arbitrary layup. *J Reinf Plast Compos* 2002; 21: 1423–1465.

<sup>4</sup> Megson TH. *Aircraft structures for engineering students*. Butterworth-Heinemann; 2016 Oct 17.

## References

# 1.5 Python API Reference

## 1.5.1 Submodules

- *abdbeam.core module*
- *abdbeam.materials module*
- *abdbeam.plots module*

### ***abdbeam.core* module**

Module defining the classes `Section`, `Segment`, `Point` and `Load`.

### **Section Class**

**class** `abdbeam.core.Section`

Class that defines a cross section, calculates its properties and internal loads.

**materials**

*dict* – Of the form {int : `abdbeam.Material`}

**points**

*dict* – Of the form {int : `abdbeam.Point`}

**segments**

*dict* – Of the form {int : `abdbeam.Segment`}

**loads**

*dict* – Of the form {int : `abdbeam.Load`}

**cells**

*dict* – An output of the form {int : `abdbeam.Cell`}

**yc**

*float* – The centroid Y coordinate.

**zc**

*float* – The centroid Z coordinate.

**ys**

*float* – The shear center Y coordinate.

**zs**

*float* – The shear center Z coordinate.

**p\_c**

*numpy.ndarray* – The 4x4 section stiffness matrix relative to the centroid.

**w\_c**

*numpy.ndarray* – The 4x4 section compliance matrix relative to the centroid.

**p**

*numpy.ndarray* – The 4x4 section stiffness matrix relative to the section origin.

**w***numpy.ndarray* – The 4x4 section compliance matrix relative to the section origin.**weight***float* – The section weight per unit length.**principal\_axis\_angle***float* – The angle of the coordinate system Y'-Z' relative to Y-Z in which the moment of Inertia  $I_{y'z'}$  is zero. Only applicable to isotropic beams.**sc\_int\_strains\_df***pandas.DataFrame* – The pandas dataframe containing the axial strain, the Y curvature, the Z curvature and the rate of twist of the section relative to the centroid (to be implemented).**sgs\_int\_lds\_df***pandas.DataFrame* – A pandas dataframe containing the segments internal loads for all load cases in the loads dictionary. Populated by the `calculate_internal_loads` method.**pts\_int\_lds\_df***pandas.DataFrame* – A pandas dataframe containing the points internal loads for all load cases in the loads dictionary. Populated by the `calculate_internal_loads` method.**summary()**

Prints a summary of the section properties.

**calculate\_properties()**

Calculates the section properties.

**calculate\_internal\_loads()**

Calculates internal loads for all load cases in the loads dictionary.

**print\_internal\_loads()**

Prints to the console segment and point internal loads for all load cases in the loads dictionary.

## Examples

Creating a 2-cells beam cross section comprised of asymmetric laminate segments (see appendix example in reference theory paper):

```

import abdbeam as ab
sc = ab.Section()
mts = dict()
mts[1] = ab.Laminate()
ply_mat = ab.PlyMaterial(0.166666, 148000, 9650, 4550, 0.3)
mts[1].ply_materials[1] = ply_mat
mts[1].plies = [[0,1], [0,1], [0,1], [0,1], [0,1], [0,1]] + [[45,1]]*6
mts[1].symmetry = 'T'
mts[1].calculate_properties()
pts = dict()
pts[1] = ab.Point(0, -35)
pts[2] = ab.Point(-50, -35)
pts[3] = ab.Point(-50, 35)
pts[4] = ab.Point(0, 35)
pts[5] = ab.Point(50, 35)
pts[6] = ab.Point(50, -35)
sgs = dict()
sgs[1] = ab.Segment(1,2,1)
sgs[2] = ab.Segment(2,3,1)

```

(continues on next page)

(continued from previous page)

```

sgs[3] = ab.Segment(3,4,1)
sgs[4] = ab.Segment(4,1,1)
sgs[5] = ab.Segment(4,5,1)
sgs[6] = ab.Segment(5,6,1)
sgs[7] = ab.Segment(6,1,1)
sc.materials = mts
sc.points = pts
sc.segments = sgs
sc.calculate_properties()
sc.summary()

```

Adding two load cases to the section above and printing their internal loads:

```

Lds = dict()
Lds[101] = ab.Load(1000.0,25000,-36000)
Lds[102] = ab.Load(Px=1500.0)
Lds[103] = ab.Load(Vz_s=1000.0)
sc.loads = Lds
sc.calculate_internal_loads()
sc.print_internal_loads()

```

#### **calculate\_internal\_loads()**

Calculates internal loads for all load cases in the loads dictionary.

Results are loaded into two pandas dataframes: `self.sgs_int_lds_df` and `self.pts_int_lds_df`. Segment loads are represented as quadratic equations by outputting the coefficients C2, C1 and C0, where  $\text{Load} = C2*n**2 + C1*n + C0$ . “n” is the location in the segment length varying from 0.0 (point A) to 1.0 (point B). Maximum and minimum segment values and their associated locations (0.0 - 1.0) inside the segment are also provided, along with the segment average and total (integrated) load.

#### **calculate\_properties()**

Calculates the section properties.

#### **print\_internal\_loads(break\_columns=True)**

Prints to the console segment and point internal loads for all load cases in the loads dictionary.

**Warning:** This method outputs a significant amount of data per load case and segment. Depending on your number of segments and load cases, manipulate the data stored in `self.sgs_int_lds_df` and `self.pts_int_lds_df` using pandas methods directly.

**Parameters `break_columns`** (*bool, default True*) – Of the form {int : abdbeam.Material}

#### **summary()**

Prints a summary of the section properties.

## Segment Class

**class** `abdbeam.core.Segment` (*point\_a\_id, point\_b\_id, material\_id, description=""*)

Class that defines a section segment and calculates its properties.

#### **point\_a\_id**

*int* – The first point id of the segment.

**point\_b\_id**  
*int* – The second point id of the segment.

**material\_id**  
*int* – The material id of the segment.

**description**  
*str* – The segment description.

**bk**  
*float* – The segment length.

**t**  
*float* – The segment thickness (based on material data).

**calculate\_properties** (*points, materials*)  
 Prints a summary of the section properties.

## Examples

Creating 3 segments and associating them to a section.

```
import abdbeam as ab
sc = ab.Section()
sgs = dict()
sgs[1] = ab.Segment(1, 2, 1)
sgs[2] = ab.Segment(2, 3, 1)
sgs[3] = ab.Segment(3, 4, 1)
sc.segments = sgs
```

**calculate\_properties** (*points, materials*)  
 Calculates the segment properties.

This method is normally called by a Section object.

### Parameters

- **points** (*dict*) – Of the form {int : abdbeam.Point}.
- **materials** (*dict*) – Of the form {int : abdbeam.Material}.

## Point Class

**class** abdbeam.core.**Point** (*y=0.0, z=0.0, EA=0.0, GJ=0.0, description=""*)  
 A cross section point in the section (Y,Z) coordinate system, optionally having an EA and GJ associated to it.

**Y**  
*float* – The Y location of the point.

**z**  
*float* – The Z location of the point.

**EA**  
*float* – The axial stiffness of the point.

**GJ**  
*float* – The torsional stiffness of the point.

**description**  
*str* – The point description.

## Examples

Create two points and associate them to a section:

```
import abdbeam as ab
sc = ab.Section()
pts = dict()
pts[1] = ab.Point(0.5, 1.0, 10000000.0, 4500000.0, 'Stringer 105')
pts[2] = ab.Point(0.0, 0.0)
sc.points = pts
```

## Load Class

```
class abdbeam.core.Load(Px_c=0.0, My=0.0, Mz=0.0, Tx=0.0, Vy_s=0.0, Vz_s=0.0, Px=0.0,
                        yp=0.0, zp=0.0, Vy=0.0, Vz=0.0, yv=0.0, zv=0.0)
```

A single section load case.

### **Px\_c**

*float* – The axial load at the centroid of the cross section. Positive sign induces tension in the cross section.

### **My**

*float* – The moment around the Y axis. Positive sign induces tension in the positive yz quadrant of the beam cross section.

### **Mz**

*float* – The moment around the Z axis. Positive sign induces tension in the positive yz quadrant of the beam cross section.

### **Tx**

*float* – The torque around the X axis. Positive sign is counterclockwise.

### **Vy\_s**

*float* – The shear force oriented with the Y axis at the shear center.

### **Vz\_s**

*float* – The shear force oriented with the section Z axis at the shear.

### **Px**

*float* – The axial force located at (yp, zp). Positive sign induces tension in the cross section.

### **yp**

*float* – The Y axis location of the Px axial force.

### **zp**

*float* – The Z axis location of the Px axial force.

### **Vy**

*float* – The shear force oriented with the Y axis at zv.

### **Vz**

*float* – The shear force oriented with the Z axis at yv.

### **yv**

*float* – The Y axis location of the Vz shear force.

### **zv**

*float* – The Z axis location of the Vy shear force.



## Examples

Creating 3 load cases and associating them to a section:

```
import abdbeam as ab
sc = ab.Section()
Lds = dict()
Lds[101] = ab.Load(My=5e6)
Lds[102] = ab.Load(Tx=250000, Vz=5000.0)
Lds[103] = ab.Load(0, 0, 0, 0, 0, 1000.0)
sc.loads = Lds
```

## abdbeam.materials module

Contains all material classes used by Section objects.

### Material Class

**class** abdbeam.materials.**Material** (*t*, *abd\_c*=<sphinx.ext.autodoc.importer.\_MockObject object>, *description*=")

Bases: object

Parent class for all materials.

May be instantiated directly but self.abd\_c needs to be manually entered.

**t**

*float* – The thickness of the material. Since for this parent class the compliance matrix is provided directly, the thickness is used for reference/plot purposes only.

**abd\_c**

*numpy.ndarray* – The material 6x6 compliance matrix based on CLT (Classical Laminate Theory).

**abd**

*numpy.ndarray* – The material 6x6 stiffness matrix based on CLT (Classical Laminate Theory).

**description**

*str* – The description of the material.

**calculate\_properties** ()

Method used by classes that inherit this base class to calculate the stiffness abd and compliance abd\_c matrices of the material based on the Classical Laminate Theory.

**calculate\_properties** ()

Method used by classes that inherit this base class to calculate the stiffness abd and compliance abd\_c matrices of the material based on the Classical Laminate Theory.

For this parent class, abd\_c needs to be manually provided.

### Isotropic Class

**class** abdbeam.materials.**Isotropic** (*t*, *E*, *v*, *description*=")

Bases: *abdbeam.materials.Material*

An isotropic material that inherits the Material class.

**t**  
*float* – The thickness of the material.

**E**  
*float* – The Young Modulus of the material.

**v**  
*float* – The Poisson Ratio of the material.

**description**  
*str* – The description of the material.

**abd\_c**  
*numpy.ndarray* – The material 6x6 compliance matrix based on CLT (Classical Laminate Theory).

**abd**  
*numpy.ndarray* – The material 6x6 stiffness matrix based on CLT (Classical Laminate Theory).

**calculate\_properties()**  
Calculates the stiffness abd and compliance abd\_c matrices of the isotropic material based on the Classical Laminate Theory.

## Examples

```
mts = dict()
mts[1] = ab.Isotropic(0.08, 10600000, 0.33)
mts[1].calculate_properties()
```

**calculate\_properties()**  
Calculates the stiffness abd and compliance abd\_c matrices of the isotropic material based on the Classical Laminate Theory.

## Laminate Class

**class** `abdbeam.materials.Laminate`  
Bases: `abdbeam.materials.Material`

A composite laminate material that inherits the Materials class.

**t**  
*float* – The laminate thickness. Calculated by the `calculate_properties()` method.

**ply\_materials**  
*dict* – Of the form {int : `abdbeam.PlyMaterial`}

**plies**  
*list* – A list that defines the laminate stacking sequence. Plies are the elements of this list, which in turn are represented as 2-elements lists of angle and material ids the form [float, int]. The first element in the plies list is the bottom ply.

**symmetry**  
{‘T’, ‘S’, ‘SM’, ‘SMEAR’}, *default* = ‘T’ – ‘T’ means all plies are defined in the plies list; ‘S’ means symmetry will be applied to the plies list; ‘SM’ means symmetry will be applied around the last item in the plies list; ‘SMEAR’ means the effects of the plies stacking sequence will be ignored ([D]=t\*\*2 / 12 \* [A]).

**abd\_c**  
*numpy.ndarray* – The material 6x6 compliance matrix based on CLT (Classical Laminate Theory).

**abd**

*numpy.ndarray* – The material 6x6 stiffness matrix based on CLT (Classical Laminate Theory).

**calculate\_properties()**

Calculates the stiffness abd and compliance abd\_c matrices of the laminate based on the Classical Laminate Theory.

**Examples**

Creating a symmetric and balanced 8 plies laminate:

```
mts = dict()
mts[1] = ab.Laminate()
ply_mat = ab.PlyMaterial(0.166666, 148000, 9650, 4550, 0.3)
mts[1].ply_materials[1] = ply_mat
mts[1].plies = [[45,1], [-45,1], [0,1], [90,1]]
mts[1].symmetry = 'S'
mts[1].calculate_properties()
```

**calculate\_properties()**

Calculates the stiffness abd and compliance abd\_c matrices of the laminate based on the Classical Laminate Theory.

**PlyMaterial Class**

**class** abdbeam.materials.**PlyMaterial** (*t, E1, E2, G12, v12, description=""*)

Bases: object

A ply material used by the Laminate class.

**t**

*float* – The ply thickness.

**E1**

*float* – The axial stiffness of the ply.

**E2**

*float* – The transverse stiffness of the ply.

**G12**

*float* – The shear modulus of the ply.

**description**

*str* – The description of the ply.

**Examples**

```
ply_mat = ab.PlyMaterial(0.166666, 148000, 9650, 4550, 0.3)
mts[1].ply_materials[1] = ply_mat
```

**Laminate Class**

**class** abdbeam.materials.**PlyMaterial** (*t, E1, E2, G12, v12, description=""*)

Bases: object

A ply material used by the Laminate class.

**t**

*float* – The ply thickness.

**E1**

*float* – The axial stiffness of the ply.

**E2**

*float* – The transverse stiffness of the ply.

**G12**

*float* – The shear modulus of the ply.

**description**

*str* – The description of the ply.

## Examples

```
ply_mat = ab.PlyMaterial(0.166666, 148000, 9650, 4550, 0.3)
mts[1].ply_materials[1] = ply_mat
```

## Shear Connector Class

**class** `abdbeam.materials.ShearConnector` (*t*, *G*, *description*=")

Bases: `abdbeam.materials.Material`

A shear connector that inherits the Material class.

Shear connectors have only the stiffness term A66 as nonzero and for this reason can only transfer shear loads. Small values are added to other terms of the stiffness matrix to prevent a singular matrix. For this material, only the compliance term a66 becomes an user input and is equal to  $1/(G*t)$ .

**t**

*float* – The thickness of the shear connector material.

**G**

*float* – The Shear Modulus of the shear connector material.

**description**

*str* – The description of the material.

**abd\_c**

*numpy.ndarray* – The material 6x6 compliance matrix based on CLT (Classical Laminate Theory).

**abd**

*numpy.ndarray* – The material 6x6 stiffness matrix based on CLT (Classical Laminate Theory).

**calculate\_properties** ()

Calculates the stiffness abd and compliance abd\_c matrices of the shear connector based on the Classical Laminate Theory.

## Examples

```
mts = dict()
mts[1] = ab.ShearConnector(0.075, 6380000, 'Shear-only Skin')
mts[1].calculate_properties()
```

### **calculate\_properties()**

Calculates the stiffness abd and compliance abd\_c matrices of the shear connector based on the Classical Laminate Theory.

## **abdbeam.plots module**

Module containing methods to plot cross sections and their internal loads.

```
abdbeam.plots.plot_section(section, segment_coord=False, thickness=True, mid_plane=True,
                           top_bottom=False, centroid=True, shear_center=True, origin=True,
                           princ_dir=True, show_axis=True, prop_color='r', pt_size=4, filter_sgs=[],
                           plot_sgs=[], legend=True, title="", figsize=(6.4, 4.8),
                           dpi=80)
```

Uses matplotlib to plot the section geometry and its properties (centroid, shear center and principal axis).

---

**Note:** Section properties need to be calculated using the method `abdbeam.Section.calculate_properties()` before using this function.

---

### **Parameters**

- **section** (*abdbeam.Section*) – The section object to be plotted.
- **segment\_coord** (*bool, default False*) – If True, will plot the segments local coordinate systems.
- **thickness** (*bool, default True*) – If True, will plot the segments thickness.
- **mid\_plane** (*bool, default True*) – If True, will plot the segments mid-plane.
- **top\_bottom** (*bool, default False*) – If True and thickness is also True, will identify the bottom side of the material with the red color and the top side with the green color.
- **centroid** (*bool, default True*) – If True, will plot the centroid location with an 'o' marker.
- **shear\_center** (*bool, default True*) – If True, will plot the shear center location with a 'x' marker.
- **origin** (*bool, default True*) – If True, will plot the section origin location with a '+' marker.
- **princ\_dir** (*bool, default True*) – If True, will plot the moment of inertia principal axes.
- **show\_axis** (*bool, default True*) – If True, will show plot dimensions.
- **prop\_color** (*string, default 'r'*) – The matplotlib color name to be used when plotting centroid, shear center and principal axes.
- **pt\_size** (*int, default 4*) – The size in pixels of the marker. Booms (points with EAs and GJs) will have 2 times this size.
- **filter\_sgs** (*list, default []*) – The list of segment ids that will not be plotted. Of the form [int]. Will also respect the filter imposed by the `plot_sgs` parameter.

- **plot\_sgs** (*list*, *default []*) – The list containing the only segment ids that will be plotted. Of the form [int]. If left empty will plot all segments. Will also respect the filter imposed by the filter\_sgs parameter.
- **legend** (*bool*, *default True*) – If True, will show a legend for the centroid, shear center and principal axis.
- **title** (*str*, *default ''*) – The title to be added at the top of the figure.
- **figsize** (*tuple*, *default (6.4, 4.8)*) – Width and height of the figure in inches. Of the form (float, float).
- **dpi** (*integer*, *default 80*) – The resolution of the figure.

## Examples

The example below creates a “C” section and plots it. The optional attribute prop\_color is changed to purple usinh the HTML color code #800080:

```
import abdbeam as ab
sc = ab.Section()
mts = dict()
mts[1] = ab.Isotropic(0.08, 10600000, 0.33)
pts = dict()
pts[1] = ab.Point(0, -1.5)
pts[2] = ab.Point(-1, -1.5)
pts[3] = ab.Point(-1, 1.5)
pts[4] = ab.Point(0.5, 1.5)
sgs = dict()
sgs[1] = ab.Segment(1,2,1)
sgs[2] = ab.Segment(2,3,1)
sgs[3] = ab.Segment(3,4,1)
sc.materials = mts
sc.points = pts
sc.segments = sgs
sc.calculate_properties()
ab.plot_section(sc, prop_color='#800080')
```

```
abdbeam.plots.plot_section_loads(section, load_id, int_load_list=['Nx', 'Nxy',
'Mx', 'My', 'Mxy'], title_list=[], thickness=True,
pt_size=4, segment_contour=True, diagram=True,
diagram_contour=False, diagram_alpha=0.15, di-
agram_scale=1.0, diagram_factor_list=[], con-
tour_color='jet_r', contour_levels=10, filter_sgs=[],
plot_sgs=[], no_result_sgs=[], result_sgs=[], figsize=(6.4,
4.8), dpi=80)
```

Uses matplotlib to plot the internal loads associated to a section and load case id.

---

**Note:** Internal loads need to be calculated using the method `abdbeam.Section.calculate_internal_loads()` before using this function; If two or more internal loads are plotted, the plots will be presented in two columns; figure sizes are for individual plots and not the entire figure.

---

## Parameters

- **section** (*abdbeam.Section*) – The section object to be plotted.

- **load\_id**(*int*) – The load case id key in the `abdbeam.Section.loads` dictionary.
- **int\_load\_list**(*list*, *default* [`'Nx'`, `'Nxy'`, `'Mx'`, `'My'`, `'Mxy'`]) – The internal load type list to be plotted for the selected load case.
- **title\_list**(*list*, *default* []) – A list containing all the plot titles to be added. An empty list (the default) will use the list `int_load_list` as titles. If the length of this list is smaller than `int_load_list`'s length, `None` values will be assumed for the last items.
- **thickness**(*bool*, *default* `True`) – If `True`, will plot the segments thickness.
- **pt\_size**(*int*, *default* `4`) – The size in pixels of the marker. Booms (points with EAs and GJs) will have 2 times this size.
- **segment\_contour**(*bool*, *default* `True`) – If `True`, will plot the internal load contour inside a segment thickness.
- **diagram**(*bool*, *default* `True`) – If `True`, will plot internal load diagrams at each segment. Positive values are plotted towards the segment top side and negative towards the bottom side.
- **diagram\_contour**(*bool*, *default* `False`) – If `True`, will replace the standard gray diagram color, with each segments' result contour.
- **diagram\_alpha**(*float*, *default* `0.15`) – The diagram transparency alpha.
- **diagram\_scale**(*float*, *default* `1.0`) – A scale factor to be applied to the diagram plot. Negative values will reverse its plot direction. Does not affect the result values, only the diagram plot.
- **diagram\_factor\_list**(*list*, *default* []) – A list containing factors to multiply each segment's diagram. An empty list (the default) is a list with factors=1. If the length of this list is smaller than the number of segments, 1.0 values will be assumed for the last items. The factors' order is the same as the order in which the segments were entered in the section segments dictionary.
- **contour\_color**(*str*, *default* `'jet_r'`) – The matplotlib's colormap name to be used in all contours.
- **contour\_levels**(*int*, *default* `10`) – The number of contour level color areas to be used.
- **filter\_sgs**(*list*, *default* []) – The list of segment ids that will not be plotted. Of the form [int]. Will also respect the filter imposed by the `plot_sgs` parameter.
- **plot\_sgs**(*list*, *default* []) – The list containing the only segment ids that will be plotted. Of the form [int]. If left empty will plot all segments. Will also respect the filter imposed by the `filter_sgs` parameter.
- **no\_result\_sgs**(*list*, *default* []) – The list of segment ids that will not have results plotted. Of the form [int]. Will also respect the filter imposed by the `result_sgs` parameter.
- **result\_sgs**(*list*, *default* []) – The list containing the only segment ids that will have results plotted. Of the form [int]. If left empty will plot contours for all segments. Will also respect the `result_sgs`.
- **figsize**(*tuple*, *default* `(6.4, 4.8)`) – Width and height of each internal load in inches. Of the form (float, float). Note that this is not the size of the entire matplotlib figure, but the size of each internal load plot.
- **dpi**(*integer*, *default* `100`) – The resolution of the figure.

## Examples

The example below creates a “C” section, creates load case id 100 with a vertical shear at the shear center of 150 and plots the ‘Nxy’ internal loads:

```
import abdbeam as ab
sc = ab.Section()
mts = dict()
mts[1] = ab.Isotropic(0.08, 10600000, 0.33)
pts = dict()
pts[1] = ab.Point(0, -1.0)
pts[2] = ab.Point(-1, -1.0)
pts[3] = ab.Point(-1, 1.0)
pts[4] = ab.Point(0, 1.0)
sgs = dict()
sgs[1] = ab.Segment(1, 2, 1)
sgs[2] = ab.Segment(2, 3, 1)
sgs[3] = ab.Segment(3, 4, 1)
sc.materials = mts
sc.points = pts
sc.segments = sgs
sc.calculate_properties()
Lds = dict()
Lds[100] = ab.Load(Vz_s=150)
sc.loads = Lds
sc.calculate_internal_loads()
ab.plot_section_loads(sc, 100, int_load_list=['Nxy'])
```

## 1.6 License

BSD 3-Clause License

Copyright (c) 2018-2019, Danilo Seixas Victorazzo All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



## CHAPTER 2

---

### Source Code

---

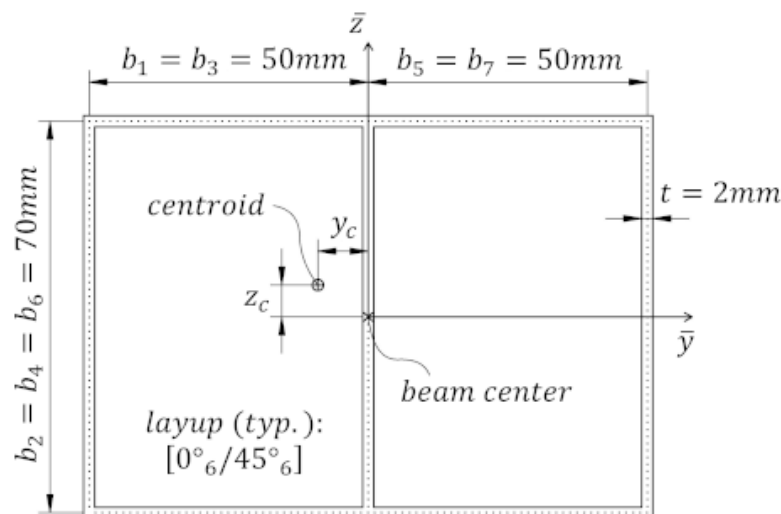
The source code is hosted on GitHub at <https://github.com/victorazzo/abdbeam>.



## CHAPTER 3

### Quick Example

Let's use *Abdbeam* to analyze the cross section with two closed cells below (from<sup>2</sup>):



Start creating the section materials, its points and segments (we'll also calculate the section properties and request a summary at the end):

```
import abdbeam as ab
sc = ab.Section()
# Create a materials dictionary:
mts = dict()
mts[1] = ab.Laminate()
ply_mat = ab.PlyMaterial(0.166666, 148000, 9650, 4550, 0.3)
mts[1].ply_materials[1] = ply_mat
mts[1].plies = [[0,1]]*6 + [[45,1]]*6
```

(continues on next page)

<sup>2</sup> Victorazzo DS, De Jesus A. A Kollár and Pluzsik anisotropic composite beam theory for arbitrary multicelled cross sections. Journal of Reinforced Plastics and Composites. 2016 Dec;35(23):1696-711.

(continued from previous page)

```

# Create a points dictionary based on Y and Z point coordinates:
pts = dict()
pts[1] = ab.Point(0, -35)
pts[2] = ab.Point(-50, -35)
pts[3] = ab.Point(-50, 35)
pts[4] = ab.Point(0, 35)
pts[5] = ab.Point(50, 35)
pts[6] = ab.Point(50, -35)
# Create a segments dictionary referencing point and material ids:
sgs = dict()
sgs[1] = ab.Segment(1,2,1)
sgs[2] = ab.Segment(2,3,1)
sgs[3] = ab.Segment(3,4,1)
sgs[4] = ab.Segment(4,1,1)
sgs[5] = ab.Segment(4,5,1)
sgs[6] = ab.Segment(5,6,1)
sgs[7] = ab.Segment(6,1,1)
# Point the dictionaries to the section
sc.materials = mts
sc.points = pts
sc.segments = sgs
# Calculate and output section properties
sc.calculate_properties()
sc.summary()

```

Which prints:

```

Section Summary
=====

Number of points: 6
Number of segments: 7
Number of cells: 2

Centroid
-----
yc      = -2.67780636e-01
zc      = 0.00000000e+00

Shear Center
-----
ys      = 2.35301214e-03
zs      = -1.45758049e-03

Replacement Stiffnesses
-----
EA       = 6.80329523e+07
EIyy     = 5.24834340e+10
EIzz     = 8.36408748e+10
EIyz     = 0.00000000e+00
GJ       = 1.23762317e+10
EImax    = 8.36408748e+10
EImin    = 5.24834340e+10
Angle    = 0.00000000e+00

[P_c] - Beam Stiffness Matrix at the Centroid
-----

```

(continues on next page)

(continued from previous page)

```
[ [ 6.80329523e+07  0.00000000e+00  2.46320132e+05 -1.43701515e+08]
  [ 0.00000000e+00  5.24834340e+10  0.00000000e+00  0.00000000e+00]
  [ 2.46320132e+05  0.00000000e+00  8.36408748e+10 -2.12142163e+07]
  [-1.43701515e+08  0.00000000e+00 -2.12142163e+07  1.23762317e+10]]
```

[W\_c] - Beam Compliance Matrix at the Centroid

```
[ [1.50683149e-08  0.00000000e+00  1.66286490e-28  1.74959530e-10]
  [0.00000000e+00  1.90536313e-11  0.00000000e+00  0.00000000e+00]
  [1.57282135e-25  0.00000000e+00  1.19558821e-11  2.04936911e-14]
  [1.74959530e-10  0.00000000e+00  2.04936911e-14  8.28315446e-11]]
```

[P] - Beam Stiffness Matrix at the Origin

```
[ [ 6.80329523e+07  0.00000000e+00 -1.79715871e+07 -1.43701515e+08]
  [ 0.00000000e+00  5.24834340e+10  0.00000000e+00  0.00000000e+00]
  [-1.79715871e+07  0.00000000e+00  8.36456213e+10  1.72662667e+07]
  [-1.43701515e+08  0.00000000e+00  1.72662667e+07  1.23762317e+10]]
```

[W] - Beam Compliance Matrix at the Origin

```
[ [1.50691722e-08  0.00000000e+00  3.20155371e-12  1.74965018e-10]
  [0.00000000e+00  1.90536313e-11  0.00000000e+00  0.00000000e+00]
  [3.20155371e-12  0.00000000e+00  1.19558821e-11  2.04936911e-14]
  [1.74965018e-10  0.00000000e+00  2.04936911e-14  8.28315446e-11]]
```

Now let's create two load cases (101 and 102) and calculate their internal loads:

```
sc.loads = dict()
sc.loads[101] = ab.Load(My=5e6)
sc.loads[102] = ab.Load(Tx=250000, Vz=5000.0)
sc.calculate_internal_loads()
```

Next print all internal loads (which outputs a lot of data we'll not show here):

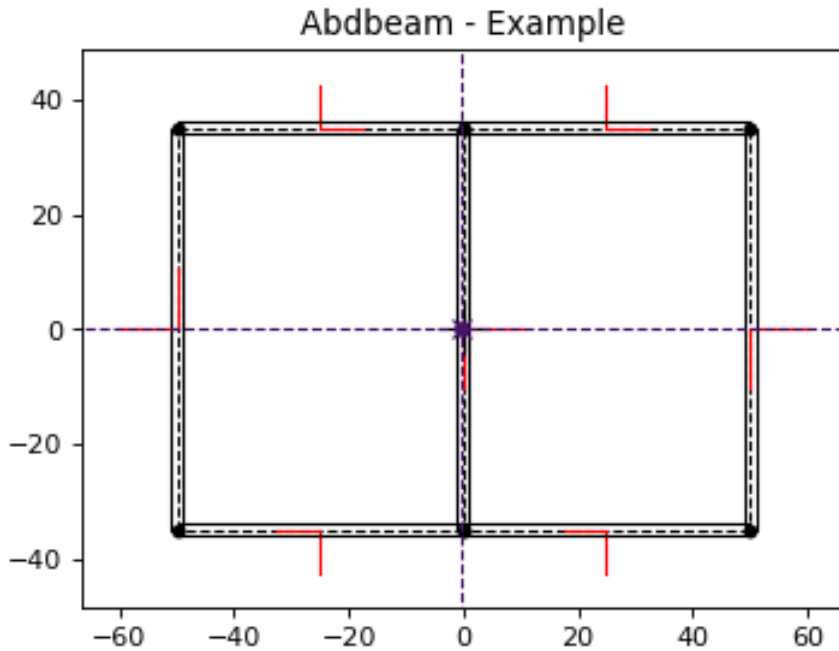
```
sc.print_internal_loads()
```

Or access the Pandas dataframe containing these internal loads directly:

```
df = sc.sgs_int_lds_df
```

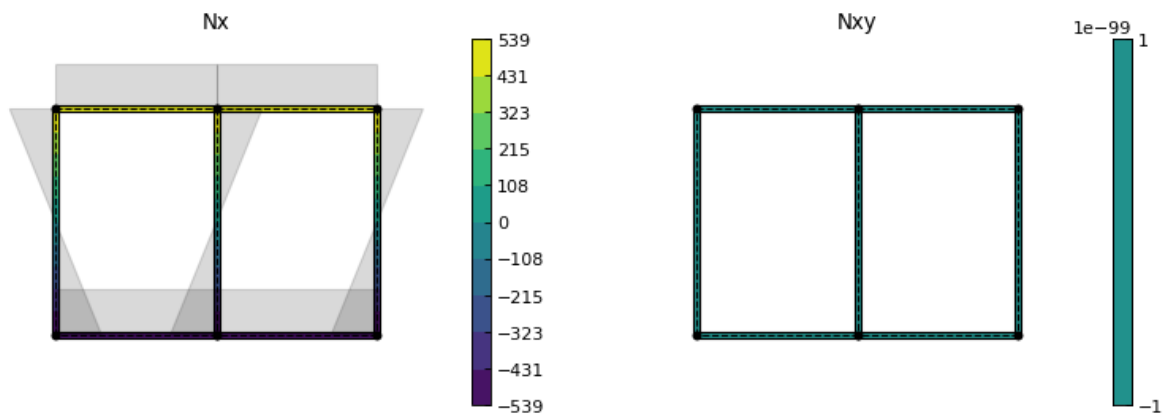
Next plot the cross section and its properties (we'll show the segment orientations, hide legends, change the centroid, shear center and principal axis colors and use a custom figure size):

```
ab.plot_section(sc, segment_coord=True, title='Abdbeam - Example',
               legend=False, prop_color='#471365', figsize=(5.12, 3.84))
```



Finally, plot  $N_x$  and  $N_{xy}$  for load case 101 (we'll change the matplotlib contour palette, reduce the internal load diagram scale, and use a custom figure size):

```
ab.plot_section_loads(sc, 101, contour_color='viridis', diagram_scale=0.7,
                     int_load_list=[' $N_x$ ', ' $N_{xy}$ '], figsize=(5.12, 3.84))
```



## CHAPTER 4

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





### **a**

`abdbeam.core`, [24](#)

`abdbeam.materials`, [29](#)

`abdbeam.plots`, [33](#)



## A

abd (abdbeam.materials.Isotropic attribute), 30  
 abd (abdbeam.materials.Laminate attribute), 30  
 abd (abdbeam.materials.Material attribute), 29  
 abd (abdbeam.materials.ShearConnector attribute), 32  
 abd\_c (abdbeam.materials.Isotropic attribute), 30  
 abd\_c (abdbeam.materials.Laminate attribute), 30  
 abd\_c (abdbeam.materials.Material attribute), 29  
 abd\_c (abdbeam.materials.ShearConnector attribute), 32  
 abdbeam.core (module), 24  
 abdbeam.materials (module), 29  
 abdbeam.plots (module), 33

## B

bk (abdbeam.core.Segment attribute), 27

## C

calculate\_internal\_loads() (abdbeam.core.Section method), 25, 26  
 calculate\_properties() (abdbeam.core.Section method), 25, 26  
 calculate\_properties() (abdbeam.core.Segment method), 27  
 calculate\_properties() (abdbeam.materials.Isotropic method), 30  
 calculate\_properties() (abdbeam.materials.Laminate method), 31  
 calculate\_properties() (abdbeam.materials.Material method), 29  
 calculate\_properties() (abdbeam.materials.ShearConnector method), 32, 33  
 cells (abdbeam.core.Section attribute), 24

## D

description (abdbeam.core.Point attribute), 27  
 description (abdbeam.core.Segment attribute), 27  
 description (abdbeam.materials.Isotropic attribute), 30  
 description (abdbeam.materials.Material attribute), 29

description (abdbeam.materials.PlyMaterial attribute), 31, 32

description (abdbeam.materials.ShearConnector attribute), 32

## E

E (abdbeam.materials.Isotropic attribute), 30  
 E1 (abdbeam.materials.PlyMaterial attribute), 31, 32  
 E2 (abdbeam.materials.PlyMaterial attribute), 31, 32  
 EA (abdbeam.core.Point attribute), 27

## G

G (abdbeam.materials.ShearConnector attribute), 32  
 G12 (abdbeam.materials.PlyMaterial attribute), 31, 32  
 GJ (abdbeam.core.Point attribute), 27

## I

Isotropic (class in abdbeam.materials), 29

## L

Laminate (class in abdbeam.materials), 30  
 Load (class in abdbeam.core), 28  
 loads (abdbeam.core.Section attribute), 24

## M

Material (class in abdbeam.materials), 29  
 material\_id (abdbeam.core.Segment attribute), 27  
 materials (abdbeam.core.Section attribute), 24  
 My (abdbeam.core.Load attribute), 28  
 Mz (abdbeam.core.Load attribute), 28

## P

p (abdbeam.core.Section attribute), 24  
 p\_c (abdbeam.core.Section attribute), 24  
 plies (abdbeam.materials.Laminate attribute), 30  
 plot\_section() (in module abdbeam.plots), 33  
 plot\_section\_loads() (in module abdbeam.plots), 34  
 ply\_materials (abdbeam.materials.Laminate attribute), 30  
 PlyMaterial (class in abdbeam.materials), 31

Point (class in abdbeam.core), 27  
point\_a\_id (abdbeam.core.Segment attribute), 26  
point\_b\_id (abdbeam.core.Segment attribute), 26  
points (abdbeam.core.Section attribute), 24  
principal\_axis\_angle (abdbeam.core.Section attribute), 25  
print\_internal\_loads() (abdbeam.core.Section method),  
25, 26  
pts\_int\_lds\_df (abdbeam.core.Section attribute), 25  
Px (abdbeam.core.Load attribute), 28  
Px\_c (abdbeam.core.Load attribute), 28  
zs (abdbeam.core.Section attribute), 24  
zv (abdbeam.core.Load attribute), 28

## S

sc\_int\_strains\_df (abdbeam.core.Section attribute), 25  
Section (class in abdbeam.core), 24  
Segment (class in abdbeam.core), 26  
segments (abdbeam.core.Section attribute), 24  
sgs\_int\_lds\_df (abdbeam.core.Section attribute), 25  
ShearConnector (class in abdbeam.materials), 32  
summary() (abdbeam.core.Section method), 25, 26  
symmetry (abdbeam.materials.Laminate attribute), 30

## T

t (abdbeam.core.Segment attribute), 27  
t (abdbeam.materials.Isotropic attribute), 29  
t (abdbeam.materials.Laminate attribute), 30  
t (abdbeam.materials.Material attribute), 29  
t (abdbeam.materials.PlyMaterial attribute), 31, 32  
t (abdbeam.materials.ShearConnector attribute), 32  
Tx (abdbeam.core.Load attribute), 28

## V

v (abdbeam.materials.Isotropic attribute), 30  
Vy (abdbeam.core.Load attribute), 28  
Vy\_s (abdbeam.core.Load attribute), 28  
Vz (abdbeam.core.Load attribute), 28  
Vz\_s (abdbeam.core.Load attribute), 28

## W

w (abdbeam.core.Section attribute), 24  
w\_c (abdbeam.core.Section attribute), 24  
weight (abdbeam.core.Section attribute), 25

## Y

y (abdbeam.core.Point attribute), 27  
yc (abdbeam.core.Section attribute), 24  
yp (abdbeam.core.Load attribute), 28  
ys (abdbeam.core.Section attribute), 24  
yv (abdbeam.core.Load attribute), 28

## Z

z (abdbeam.core.Point attribute), 27  
zc (abdbeam.core.Section attribute), 24  
zp (abdbeam.core.Load attribute), 28