IndeterminateBeam

Release 0.1

Jesse Bonanno

Jun 10, 2023

GETTING STARTED

`IndeterminateBeam` is a Python Package aiming to serve as a foundation for civil and structural engineering projects in Python. The module can also serve as a standalone program and is useful for determining:

- reaction forces for indeterminate beams
- internal forces for indeterminate beams (shear, bending, axial)
- deflection of the beam due to resulting forces
- axial force, shear force, and bending moment diagrams

The `indeterminatebeam` package repository can be found on Github and is ready for installation using *pip*.

CHAPTER

ONE

FIRST STEPS

- The easiest way to try this package is using a web-based notebook:

- You can also download and install the `indeterminatebeam` package, which runs on Python 3.
  - If you do not have a Python 3 interpreter on your machine, you can install the last version following the instructions in this tutorial.
  - Once you have Python 3, you can open a terminal and install the package with this one-liner:

```
python3 -m pip install --user indeterminatebeam
```

    NOTE: You may need to replace `python3` above by the path to your Python 3 executable, or simply `python` if you are running Windows.

This project is completely open source, and the code can be found in this GitHub repository.

CHAPTER

TWO

THEORY

A brief overview of the engineering theory and conventions used in this program are illustrated below. Theory is adapted from the Hibbeler textbook [2]. A more rigorous overview of the basic theory behind statically determinate structures is presented in the beambending package documentation here [1].

2.1 Indeterminate vs Determinate structures

The determinacy of a structure refers to its solvability mathematically with regards to equilibrium equations.

In maths, when dealing with simultaneous equations, to be able to solve for **n** number of unknowns we will also need **n** number of equations (assuming all equations are independent).

For a 1D beam in a 2D space we have 3 equilibrium equations:

1. $\sum F_x = 0$
2. $\sum F_y = 0$
3. $\sum M = 0$

To be able to solve this structure using equilibrium we can only have at most 3 unknowns. A structure which can be solved only by using these equations is referred to as *statically determinate*.

The determinacy of a structure is a metric defined as the number of unknown reaction forces - the number of equilibrium equations *available*.

For a statically determinate beam this value is 0.

There are also two more possible cases:

- The determinacy is < 0 : In this case the beam is statically *unstable*
- The determinacy is > 0 : In this case the beam is statically *indeterminate*

An *unstable* structure is one which is mathematically unsolvable. In this case our object is likely not actually going to be static as it is but will need more supports introduced.

An *indeterminate* structure is one which mathematically by equilibrium has multiple possible solutions. This is ideal in engineering since if one reaction is removed our structure still has a chance of standing up.

We can actually solve for the forces in indeterminate structures, we just have to look beyond the concept of equilibrium and look at geometrical constraints of our structure.

If you would like for a more detailed explanation of statically determinate structures refer to beambending, the python package upon which this is based upon which deals solely with the statically determinate case.

2.2 Indeterminate Calculation Theory

Since our moment equations will never help us to resolve axial forces on our 1D beam, let us consider two independent systems of indeterminacy:

1. Forces in the **x** direction
2. Forces in the **y** and **m** direction

If we have more than one unknown in the **x** direction, or more than two unknowns in the **y** and **m** directions collectively, then equilibrium will not be enough to solve for the forces on the beam.

2.2.1 Forces in the x-direction

For a single **x** restraint we can resolve our unknown force by equilibrium. When we introduce a second restraint we can use our knowledge that both of these restraints will be fixed in space, and hence the axial deformation between them will amount to 0.

- For additional supports in the x-direction we know that the displacement between these supports will be 0.

For forces in the x-direction we can get our axial displacement from:

$$e = \int_0^L \frac{N(x)}{E(x)A(x)} dx$$

Assuming E*A is constant we can simply integrate our normal force diagram between supports and the result will be 0.

$$\begin{aligned} 0 &= \int_{sup_b}^{sup_a} \frac{N(x)}{E(x)A(x)} dx \\ &= \int_{sup_b}^{sup_a} N(x) dx \end{aligned}$$

Hence for each additional support in the x direction we can have a new equation, and we can solve for all our unknowns in the x-direction.

2.2.2 Forces in the y-direction and m-direction

For supports totaling a maximum total of two unknowns in the y-direction and m-direction we can solve for our forces with equilibrium. When we introduce an additional unknown, we can use the following geometrical information to help establish additional equations:

- For additional supports in the y-direction we know that the vertical displacement will be 0.
- For additional supports in the m-direction we know that the slope of the beam will be 0.

We can get slope and vertical displacement equations by integrating the moment equation with respect to x.

$$EIv''(x) = M$$

$$EIv'(x) = \int M(x) dx + C_1$$

$$EIv(x) = \iint M(x) dx + C_1 * x + C_2$$

This introduces two new unknowns, C_1 and C_2, which are the constants for integration. Luckily, we still have our two equilibrium equations so we can solve for the integration constants.

In order to establish the internal moment equation $M(x)$ as a single equation piecewise functions are a helpful mathematical tool and are used frequently in the python package solution.

2.3 Spring Supports

Let me now introduce Hooke's Law, which states that a spring will deform proportionally to a force applied to it based on its stiffness. Mathematically this is expressed as:

$$F = -k * u$$

where k is a constant value that represents the stiffness of the spring.

Okay so in our solution for indeterminate beams we used the geometrical constraint that the displacement of our supports (u) will be 0. No matter what force the displacement will be 0, and so by Hooke's law we can see that we have idealised that our support has infinite stiffness.

On the other extreme if we said our spring had 0 stiffness i.e.. k = 0, then no matter how much the beam deflects at that support it will not resist any force. I.e.. there is no reaction, and the support does not actually exist, at least, as a support.

These are two extreme cases but what if we want to simulate a realistic value, one which isn't approaching some extreme case?

Well then we can reconsider our previous geometric constraints:

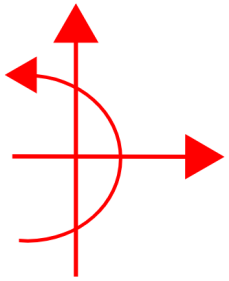
- For additional supports in the y-direction we know that the vertical displacement will be F_y/k_y
- For additional supports in the x-direction we know that the displacement between these supports will be $F_{x2}/k_{x2} - F_{x1}/K_{x1}$

We have not added any more unknowns, we still have the same equations only with a new term within. Hence our indeterminate solution is still perfectly solvable. Unlike before however, our bending rigidity EI will now affect our reaction forces in our y-m solution and our axial stiffness EA will now our affect our results for our x-force solution.

2.4 Sign Convention

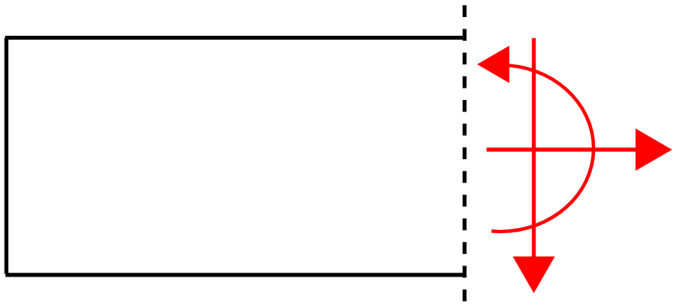
For External Forces the following convention is used:

- For x direction: To the right is positive
- For y direction: Up is positive
- For m direction: Anti-clockwise is positive



For internal forces considering the left of a cut:

- For axial force (x direction): To the right is positive
- For shear force (y direction): Down is positive
- For moments: Anti-clockwise is positive

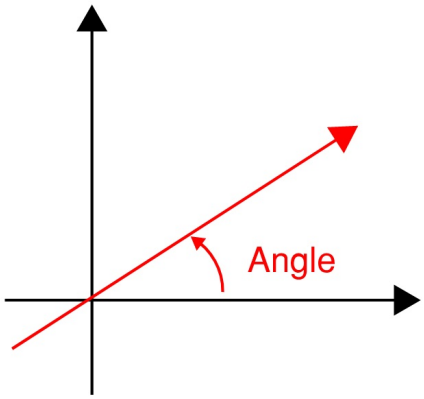


For deflections:

- Up is considered positive

For angled point loads (assuming a positive force is used):

- An angle of 0 indicates a positive force to the right
- An angle between 0 and 90 indicates a positive force to the right and a positive force up
- An angle of 90 indicates a positive force up
- An angle between 90 and 180 degrees indicates a force acting left (negative direction) and a positive force acting up
- An angle of 180 indicates a negative horizontal force



2.5 Unit Convention

The units used throughout the python package are the base SI Units. The following units are adopted in their respective sections of the application.

- The default units for length, force and bending moment (torque) are in N and m (m, N, N·m)
- The default units for beam properties (E, I, A) are in N and m (N/m2, m4, m2)
- The default unit for spring stiffness and distributed loads is N/m

Units can be updated on a beam by using the update_units() method. Adopting this method effects the way input units are considered in calculations and presents graphs dynamically based on the units that have been specified

For example, if units for 'force' are changed to 'kN' then PointLoad object forces would be interpretted to have a force value in kN. The y-axis for shear force and normal force diagrams would also be in kN and reaction force and point load forces would also be presented in kN. This change would not however effect the presentation of moments which would still appear in N.m.

CHAPTER

THREE

1. INTRODUCTION TO EXAMPLES

This section demonstrates the core functionality of the `indeterminatebeam` package with examples. Examples 5, 6, 7 and 8 have been taken from the Hibbeler textbook [2].

You can follow along with examples online:

In order to effectively use the package online you should first run the cell that initialises the notebook by installing the package.

```
# RUN THIS CELL FIRST TO INITIALISE GOOGLE NOTEBOOK!!!!
!pip install indeterminatebeam
%matplotlib inline

# import beam and supports
from indeterminatebeam import Beam, Support

# import loads (all load types imported for reference)
from indeterminatebeam import (
    PointTorque,
    PointLoad,
    PointLoadV,
    PointLoadH,
    UDL,
    UDLV,
    UDLH,
    TrapezoidalLoad,
    TrapezoidalLoadV,
    TrapezoidalLoadH,
    DistributedLoad,
    DistributedLoadV,
    DistributedLoadH
)

# Note: load ending in V are vertical loads
# load ending in H are horizontal loads
# load not ending in either takes angle as an input (except torque)
```

You can also approach the problems using the web-based graphical user interface

CHAPTER

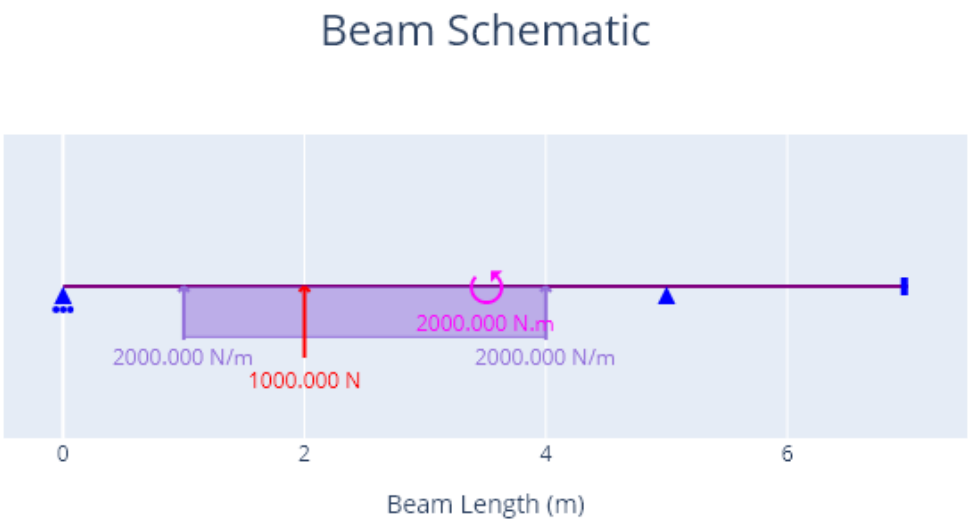
FOUR

2. BASIC USAGE (README EXAMPLE)

4.1 2.1 Basic Usage

4.1.1 Specifications

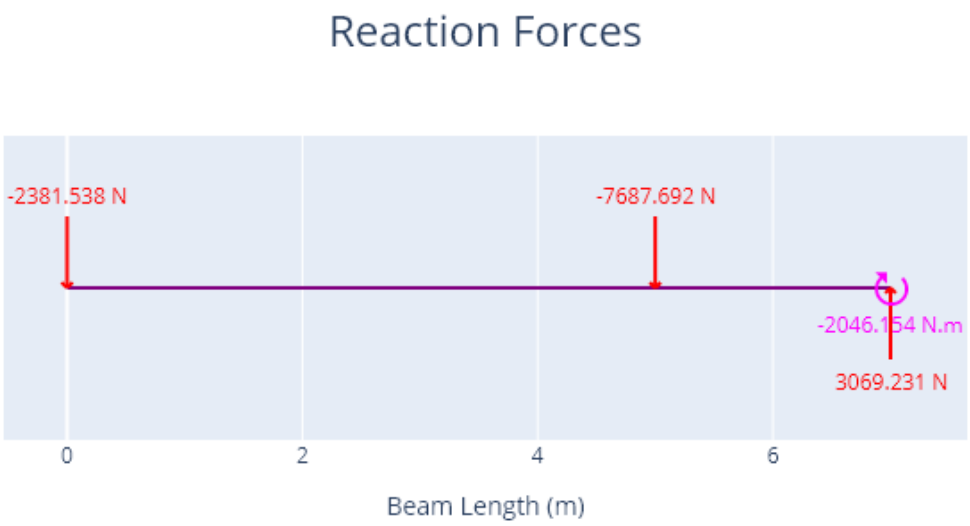
A diagram of the problem is shown below.



4.1.2 Results

A plot of the reactions is shown below.

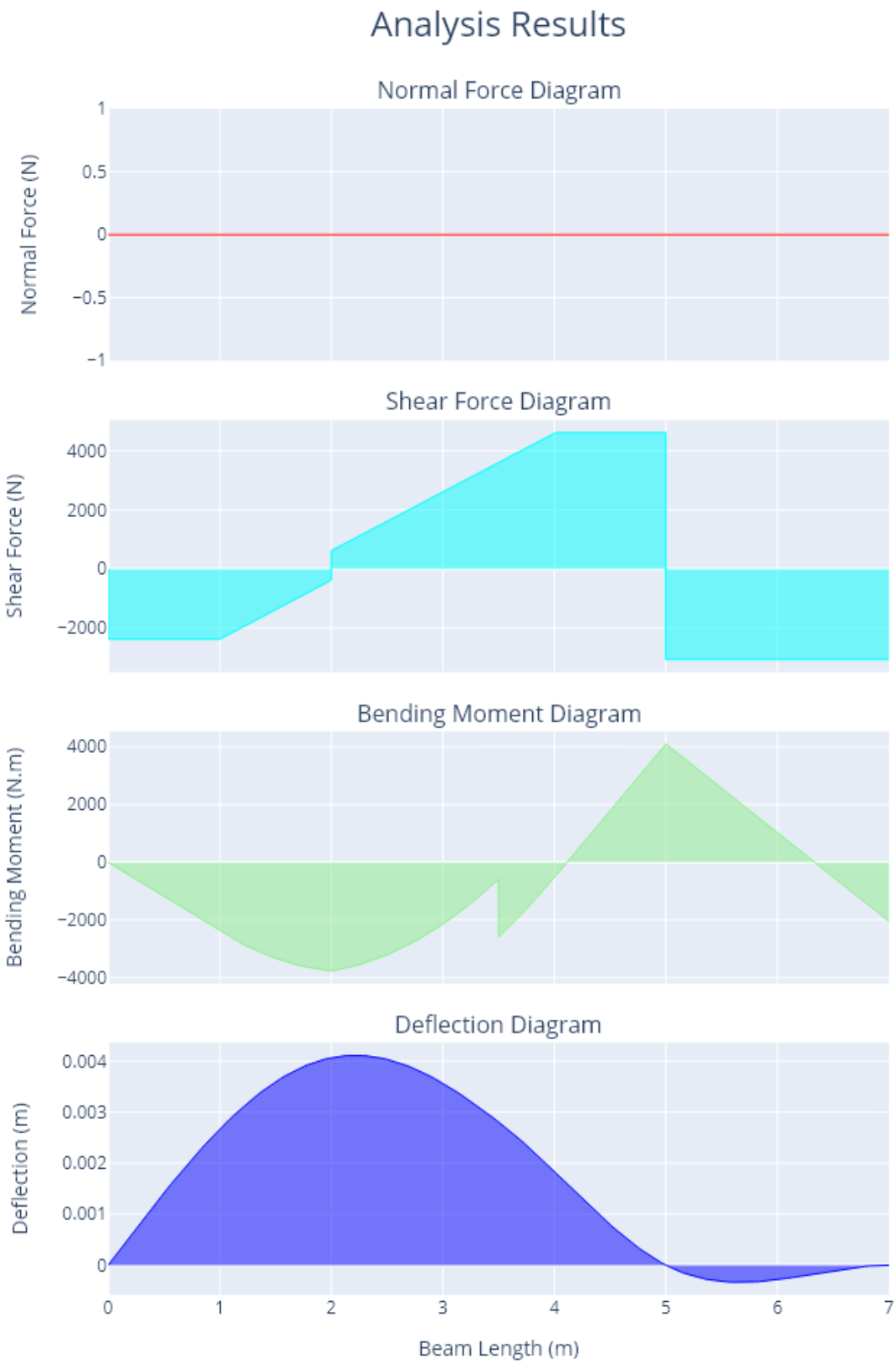
A plot of the axial force, shear force, bending moments and deflections is shown below.



4.1.3 Code

```
# Arbritrary example defined in README.md
from indeterminatebeam import Beam, Support, PointLoadV, PointTorque, UDLV,␣
↪DistributedLoadV
beam = Beam(7)                          # Initialize a Beam object of length 7 m with E␣
↪and I as defaults
beam_2 = Beam(9,E=2000, I =100000)      # Initialize a Beam specifying some beam␣
↪parameters

a = Support(5,(1,1,0))                  # Defines a pin support at location x = 5 m
b = Support(0,(0,1,0))                  # Defines a roller support at location x = 0 m
c = Support(7,(1,1,1))                  # Defines a fixed support at location x = 7 m
beam.add_supports(a,b,c)

load_1 = PointLoadV(1000,2)                # Defines a point load of 1000 N acting up,␣
↪at location x = 2 m
load_2 = DistributedLoadV(2000,(1,4))      # Defines a 2000 N/m UDL from location x = 1␣
↪m to x = 4 m
load_3 = PointTorque(2*10**3, 3.5)            # Defines a 2*10**3 N.m point torque at␣
↪location x = 3.5 m
beam.add_loads(load_1,load_2,load_3)    # Assign the support objects to a beam object␣
↪created earlier

beam.analyse()

fig_1 = beam.plot_beam_external()
fig_1.show()

fig_2 = beam.plot_beam_internal()
fig_2.show()
```

(continues on next page)


Analysis Results
Normal Force Diagram
Normal Force (N)
1
0.5
0
−0.5
−1
Shear Force Diagram
Shear Force (N)
4000
2000
0
−2000
Bending Moment Diagram
Bending Moment (N.m)
4000
2000
0
−2000
−4000
Deflection Diagram
Deflection (m)
0.004
0.003
0.002
0.001
0
0
1
2
3
4
5
6
7
Beam Length (m)

(continued from previous page)

```
# save the results (optional)
# Can save figure using ``fig.write_image("./results.pdf")`` (can change extension to be
# png, jpg, svg or other formats as reired). Requires pip install -U kaleido

# fig_1.write_image("./readme_example_diagram.png")
# fig_2.write_image("./readme_example_internal.png")
```

4.2 2.2 Querying Data

4.2.1 Specifications

The same beam as presented in 2(a) is analysed further for specific information. Querys are set at specific coordinates to allow for more precise data, rather than a general graph over view.

4.2.2 Results

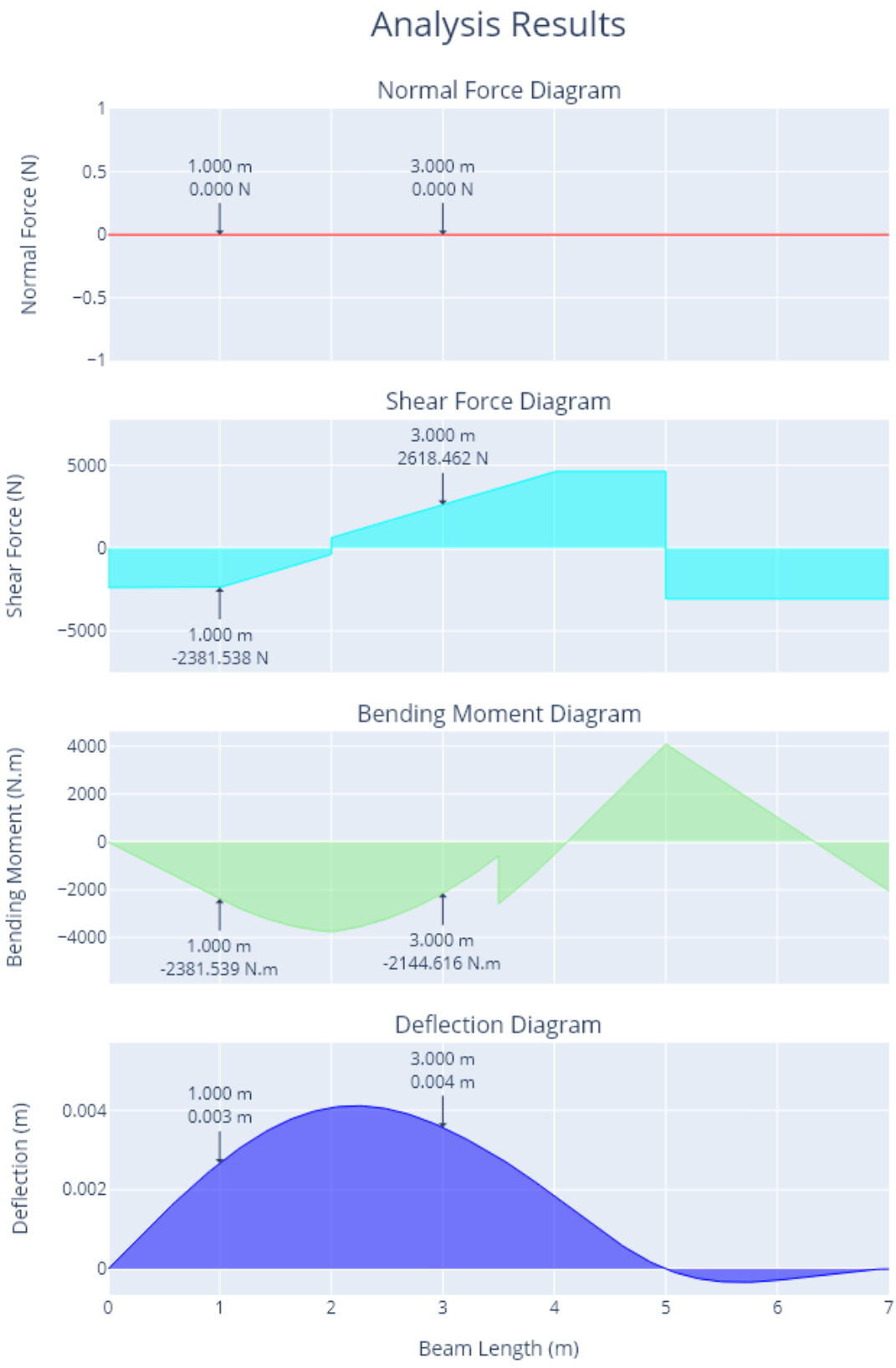
The program outputs the following text.:

```
bending moments at 3 m: -2144.6156464615
shear forces at 1,2,3,4,5m points: [-2381.5384615385, 618.4617384615, 2618.4617384615,␣
↪4618.4615384615, 4618.4615384615]
normal force absolute max: 0.0
deflection max: 0.0041098881
```

A plot of the axial force, shear force, bending moments and deflections is shown below.

4.2.3 Code

```
# Run section 2a (prior to running this example)

# query for the data at a specfic point (note units are not provided)
print("bending moments at 3 m: " + str(beam.get_bending_moment(3)))
print("shear forces at 1,2,3,4,5m points: " + str(beam.get_shear_force(1,2,3,4,5)))
print("normal force absolute max: " + str(beam.get_normal_force(return_absmax=True)))
print("deflection max: " + str(beam.get_deflection(return_max = True)))

##add a query point to a plot (adds values on plot)
beam.add_query_points(1,3,5)
beam.remove_query_points(5)

## plot the results for the beam
fig = beam.plot_beam_internal()
fig.show()
```

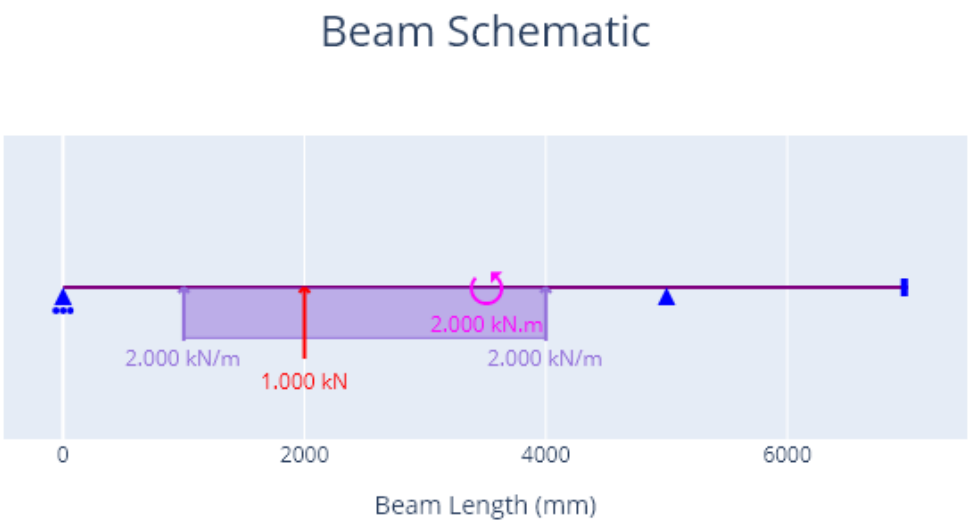

Analysis Results
Normal Force Diagram
Normal Force (N)
1.000 m
0.000 N
3.000 m
0.000 N
Shear Force Diagram
Shear Force (N)
3.000 m
2618.462 N
1.000 m
-2381.538 N
Bending Moment Diagram
Bending Moment (N.m)
1.000 m
-2381.539 N.m
3.000 m
-2144.616 N.m
Deflection Diagram
Deflection (m)
1.000 m
0.003 m
3.000 m
0.004 m
Beam Length (m)

4.3 2.3 Specifing units

4.3.1 Specifications

The same beam as presented in 2(a) is set up using alternative units for the same result. Note that units are used to modify both inputs and outputs. Default units are SI (N and m).

A diagram of the problem is shown below.



4.3.2 Results

A plot of the reactions is shown below.

A plot of the axial force, shear force, bending moments and deflections is shown below.

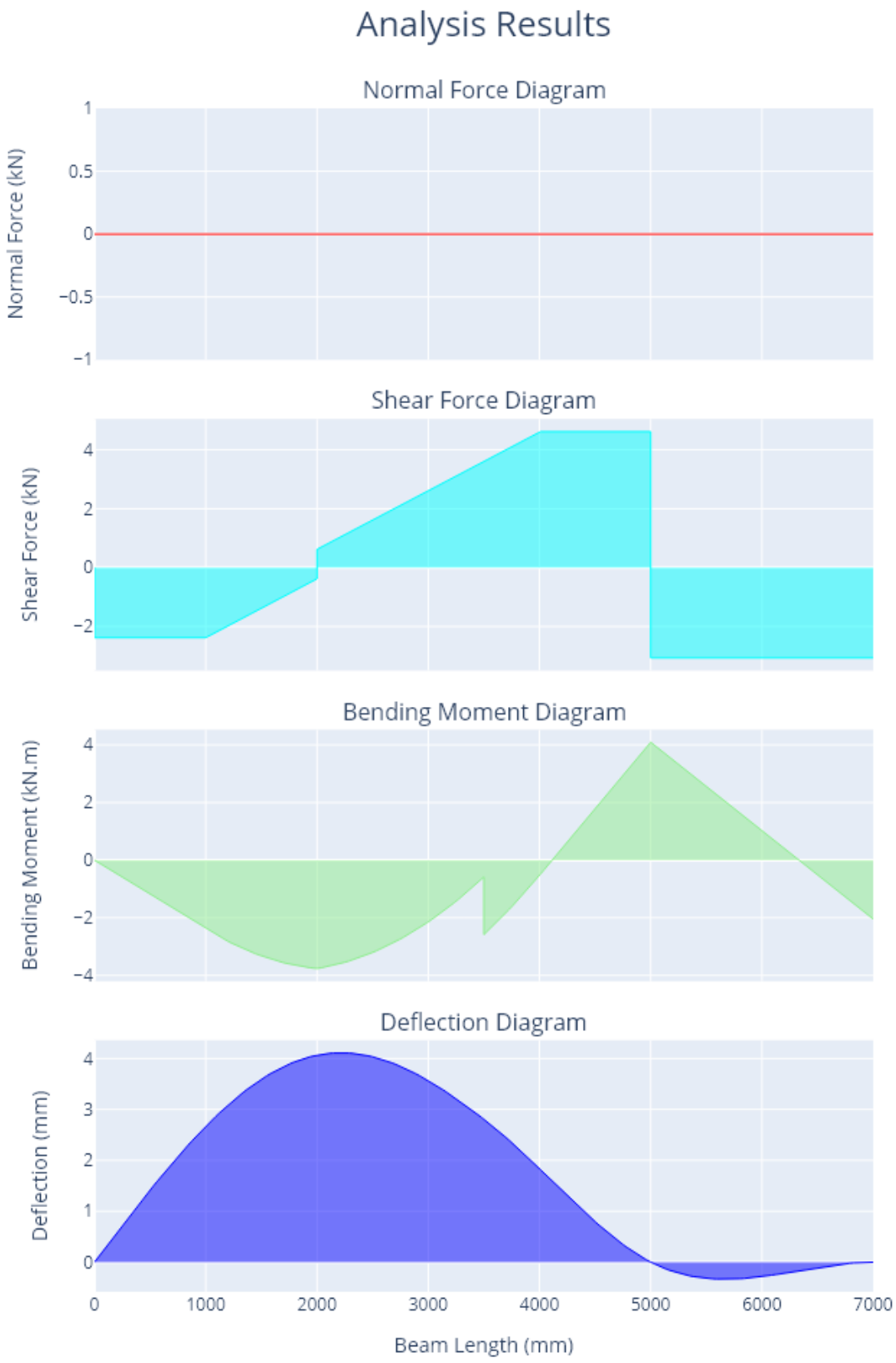
4.3.3 Code

```
# Readme example as a demonstration for changing units
# used and presented in following example. Note that
# the example is conceptually identical only with
# different units.

# update units usings the update_units() function.
# use the command below for more information.
# help(beam.update_units)

# note: initialising beam with the anticipation that units will be updated
beam = Beam(7000, E = 200 * 10 **6, I = 9.05 * 10 **6)

beam.update_units(key='length', unit='mm')
```

(continues on next page)

(continued from previous page)

```
beam.update_units('force', 'kN')
beam.update_units('distributed', 'kN/m')
beam.update_units('moment', 'kN.m')
beam.update_units('E', 'kPa')
beam.update_units('I', 'mm4')
beam.update_units('deflection', 'mm')

a = Support(5000,(1,1,0))               # Defines a pin support at location x = 5 m (x =␣
↪5000 mm)
b = Support(0,(0,1,0))                  # Defines a roller support at location x = 0 m
c = Support(7000,(1,1,1))               # Defines a fixed support at location x = 7 m (x␣
↪= 7000 mm)
beam.add_supports(a,b,c)

load_1 = PointLoadV(1,2000)             # Defines a point load of 1000 N (1 kN) acting␣
↪up, at location x = 2 m
load_2 = DistributedLoadV(2,(1000,4000))      # Defines a 2000 N/m (2 kN/m) UDL from␣
↪location x = 1 m to x = 4 m
load_3 = PointTorque(2, 3500)           # Defines a 2*10**3 N.m (2 kN.m) point torque at␣
↪location x = 3.5 m
beam.add_loads(load_1,load_2,load_3)    # Assign the support objects to a beam object␣
↪created earlier

beam.analyse()
fig_1 = beam.plot_beam_external()
fig_1.show()

fig_2 = beam.plot_beam_internal()
fig_2.show()
```


Analysis Results
Normal Force Diagram
Normal Force (kN)
1
0.5
0
−0.5
−1
Shear Force Diagram
Shear Force (kN)
4
2
0
−2
Bending Moment Diagram
Bending Moment (kN.m)
4
2
0
−2
−4
Deflection Diagram
Deflection (mm)
4
3
2
1
0
0
1000
2000
3000
4000
5000
6000
7000
Beam Length (mm)

CHAPTER

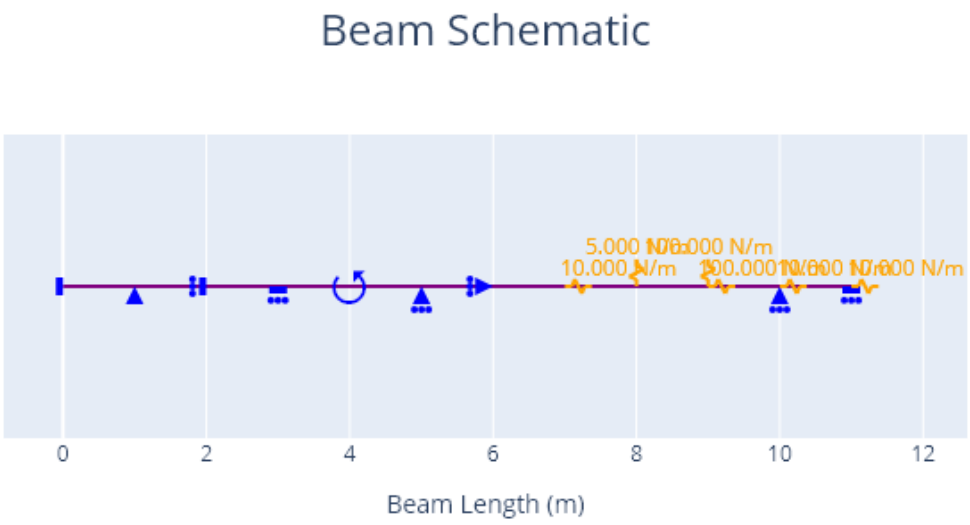
FIVE

3. SUPPORT CLASS BREAKDOWN

5.1 Representation

Supports can be created for a range of degrees of freedom. In order to ensure all possible supports are visually distinct a seperate support represenation has been used for each different possible degree of freedom combination.

The creation of different supports is shown in the code and diagram below.



5.2 Code

```
# The parameters for a support class are as below, taken from the docstring
# for the Support class __init__ method.

# Parameters:
#         -----------
#         coord: float
#             x coordinate of support on a beam in m (default 0)
#         fixed: tuple of 3 booleans
```

(continues on next page)

(continued from previous page)

```
#             Degrees of freedom that are fixed on a beam for movement in
#             x, y and bending, 1 represents fixed and 0 represents free
#             (default (1,1,1))
#         kx :
#             stiffness of x support (N/m), if set will overide the
#             value placed in the fixed tuple. (default = None)
#         ky : (positive number)
#             stiffness of y support (N/m), if set will overide the
#             value placed in the fixed tuple. (default = None)


# Lets define every possible degree of freedom combination for
# supports below, and view them on a plot:
support_0 = Support(0, (1,1,1))     # conventional fixed support
support_1 = Support(1, (1,1,0))     # conventional pin support
support_2 = Support(2, (1,0,1))
support_3 = Support(3, (0,1,1))
support_4 = Support(4, (0,0,1))
support_5 = Support(5, (0,1,0))     # conventional roller support
support_6 = Support(6, (1,0,0))

# Note we could also explicitly define parameters as follows:
support_0 = Support(coord=0, fixed=(1,1,1))

# Now lets define some spring supports
support_7 = Support(7, (0,0,0), kx = 10)    #spring in x direction only
support_8 = Support(8, (0,0,0), ky = 5)     # spring in y direction only
support_9 = Support(9, (0,0,0), kx = 100, ky = 100)     # spring in x and y direction

# Now lets define a support which is fixed in one degree of freedom
# but has a spring stiffness in another degree of freedom
support_10 = Support(10, (0,1,0), kx = 10) #spring in x direction, fixed in y direction
support_11 = Support(11, (0,1,1), kx = 10) #spring in x direction, fixed in y and m␣
↪direction

# Note we could also do the following for the same result since the spring
# stiffness overides the fixed boolean in respective directions
support_10 = Support(10, (1,1,0), kx =10)

# Now lets plot all the supports we have created
beam = Beam(11)

beam.add_supports(
    support_0,
    support_1,
    support_2,
    support_3,
    support_4,
    support_5,
    support_6,
    support_7,
    support_8,
```

(continues on next page)

(continued from previous page)

```
    support_9,
    support_10,
    support_11,
)

fig = beam.plot_beam_diagram()
fig.show()
```

CHAPTER

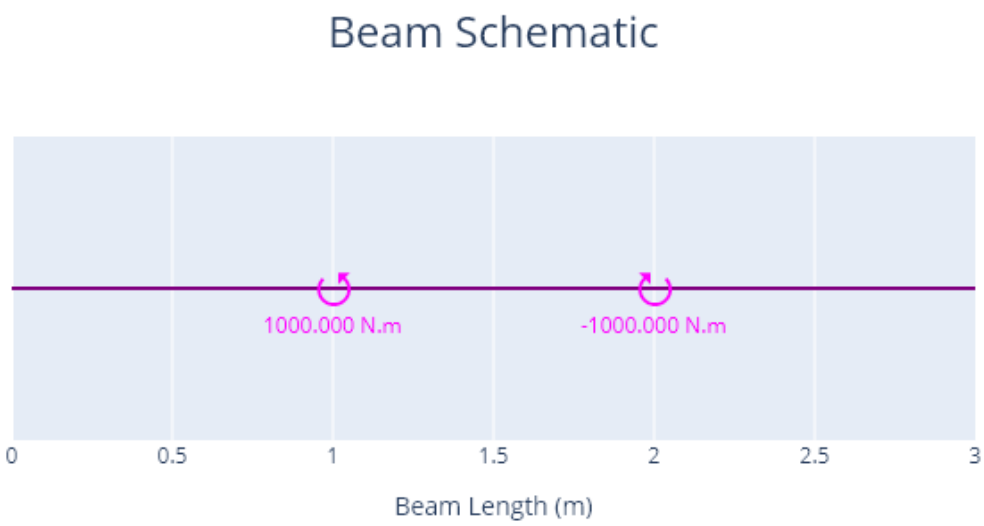
SIX

4. LOAD CLASSES BREAKDOWN

6.1 4.1 Point Torque

6.1.1 Representation

A beam loaded with point torques is shown below.



6.1.2 Code

```
# defined using a force (technically a moment, however force is used to maintain␣
↪consistenct for all load classes) and a coordinate. An anti-clockwise moment is␣
↪positive by convention of this package.
load_1 = PointTorque(force=1000, coord=1)
load_2 = PointTorque(force=-1000, coord=2)

# Plotting the loads
beam = Beam(3)
beam.add_loads(
```

(continues on next page)

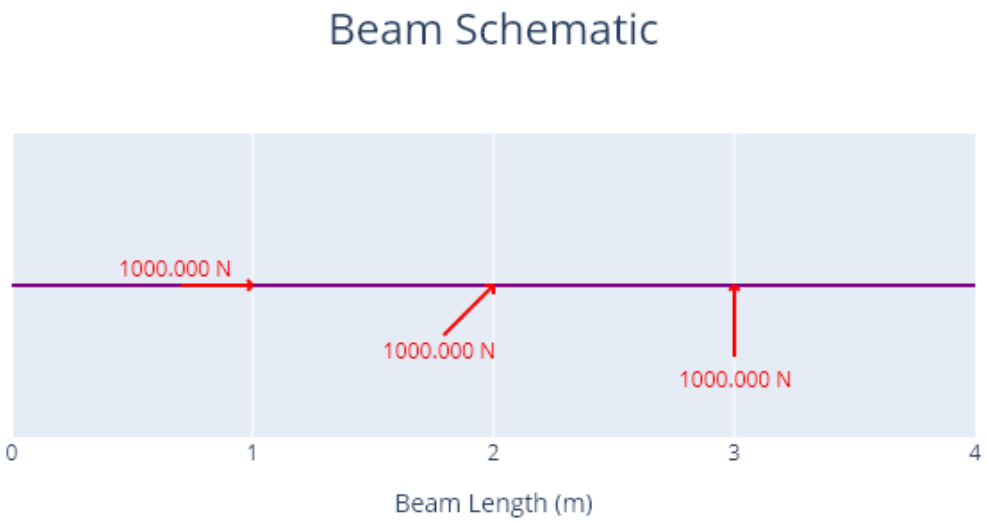
(continued from previous page)

```
    load_1,
    load_2
)

fig = beam.plot_beam_diagram()
fig.show()
```

6.2 4.2 Point Load

6.2.1 Representation

A beam loaded with point loads is shown below.



6.2.2 Code

```
# defined by force, coord and angle (0)
load_1 = PointLoad(force=1000, coord=1, angle=0)
load_2 = PointLoad(force=1000, coord=2, angle=45)
load_3 = PointLoad(force=1000, coord=3, angle=90)

# Plotting the loads
beam = Beam(4)
beam.add_loads(
    load_1,
    load_2,
    load_3
)
```

(continues on next page)

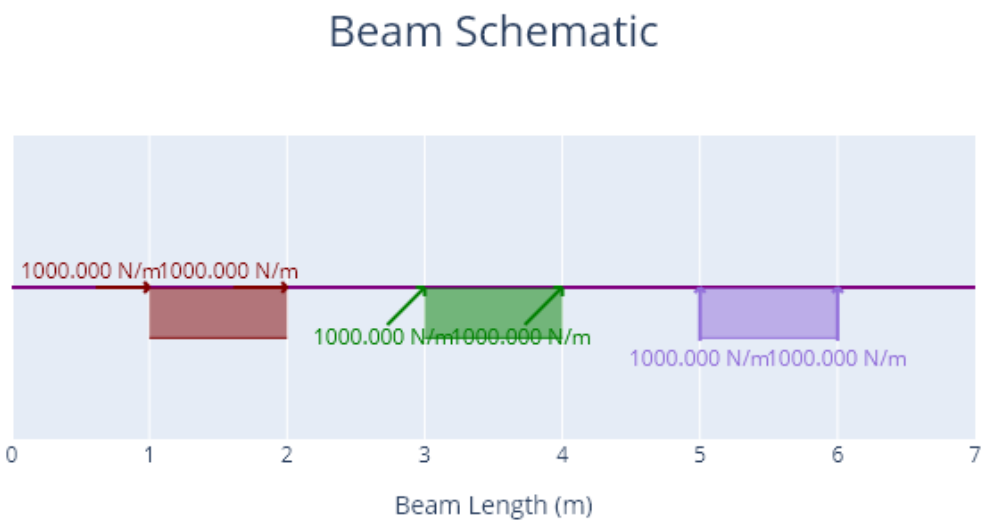
(continued from previous page)

```
fig = beam.plot_beam_diagram()
fig.show()
```

6.3 4.3 Uniformly Distributed Load (UDL)

6.3.1 Representation

A beam loaded with uniformly distributed loads is shown below.



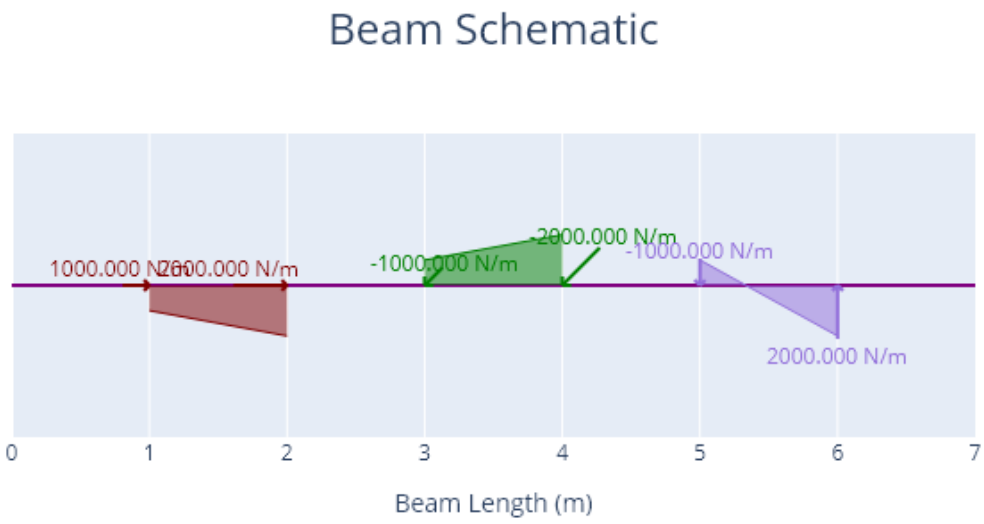
6.3.2 Code

```
# defined by force, span (tuple with start and end point)
# and angle of force
load_1 = UDL(force=1000, span=(1,2), angle = 0)
load_2 = UDL(force=1000, span=(3,4), angle = 45)
load_3 = UDL(force=1000, span=(5,6), angle = 90)

# Plotting the loads
beam = Beam(7)
beam.add_loads(
    load_1,
    load_2,
    load_3
)

fig = beam.plot_beam_diagram()
fig.show()
```

6.4 4.4 Trapezoidal Load

6.4.1 Representation

A beam loaded with trapezoidal loads is shown below.



6.4.2 Code

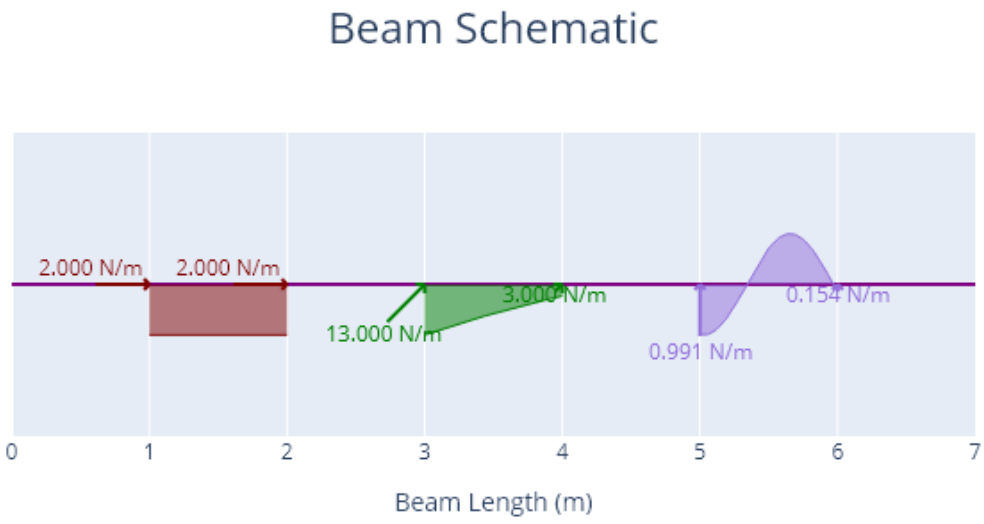
```
# defined by force (tuple with start and end force),
# span (tuple with start and end point) and angle of force
load_1 = TrapezoidalLoad(force=(1000,2000), span=(1,2), angle = 0)
load_2 = TrapezoidalLoad(force=(-1000,-2000), span=(3,4), angle = 45)
load_3 = TrapezoidalLoad(force=(-1000,2000), span=(5,6), angle = 90)

# Plotting the loads
beam = Beam(7)
beam.add_loads(
    load_1,
    load_2,
    load_3
)

fig = beam.plot_beam_diagram()
fig.show()
```

6.5 4.5 Distributed Load

6.5.1 Representation

A beam loaded with distributed loads is shown below.

It should be noted that distributed loads are treated differently by the program as compared to all other load objects. Distributed loads are set up to be more flexible but at the cost of the calculations running much slower. Where other functions can be used the distributed load type object should be avoided.



6.5.2 Code

```
# defined with Sympy expression of the distributed load function
# expressed using variable x which represents the beam x-coordinate.
# Requires quotation marks around expression. As with the UDL and
# Trapezoidal load classes other parameters to express are the span
# (tuple with start and end point) and angle of force.
# NOTE: where UDL or Trapezoidal load classes can be specified (linear functions)
# they should be used for quicker analysis times.

load_1 = DistributedLoad(expr= "2", span=(1,2), angle = 0)
load_2 = DistributedLoad(expr= "2*(x-6)**2 -5", span=(3,4), angle = 45)
load_3 = DistributedLoad(expr= "cos(5*x)", span=(5,6), angle = 90)

# Plotting the loads
beam = Beam(7)
beam.add_loads(
    load_1,
    load_2,
    load_3
)
```

(continues on next page)

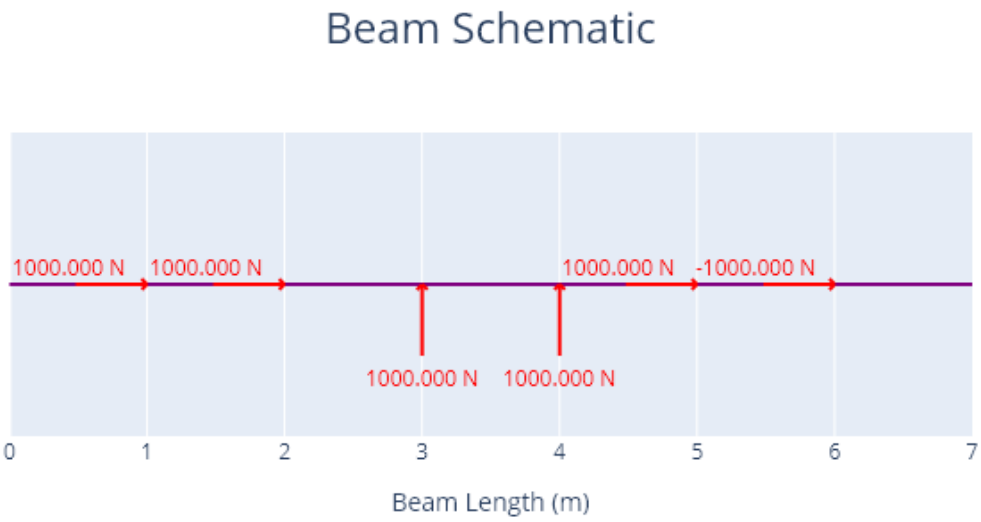
(continued from previous page)

```
fig = beam.plot_beam_diagram()
fig.show()
```

6.6 4.6 Vertical and Horizontal load child classes

6.6.1 Representation

For all loads except the point torque an angle is specified for the direction of the load. If the load to be specified is to be completely vertical or completely horizontal a V (vertical) or a H (horizontal) can be added at the end of the class name, and the angle does then not need to be specified.



6.6.2 Code

```
# for all loads except the point torque an angle is specified for the
# direction of the load. If the load to be specified is to be completely
# vertical or completely horizontal a V (vertical) or a H (horizontal)
# can be added at the end of the class name, and the angle does then
# not need to be spefied.

# The following two loads are equivalent horizontal loads
load_1 = PointLoad(force=1000, coord=1, angle = 0)
load_2 = PointLoadH(force=1000, coord=2)

# The following two loads are equivalent vertical loads
load_3 = PointLoad(force=1000, coord=3, angle = 90)
load_4 = PointLoadV(force=1000, coord=4)
```

(continues on next page)

(continued from previous page)

```
# The following two loads are also equivalent (a negative sign
# esentially changes the load direction by 180 degrees).
load_5 = PointLoad(force=1000, coord=5, angle = 0)
load_6 = PointLoad(force=-1000, coord=6, angle = 180)


# Plotting the loads
beam = Beam(7)
beam.add_loads(
    load_1,
    load_2,
    load_3,
    load_4,
    load_5,
    load_6
)

fig = beam.plot_beam_diagram()
fig.show()
```

CHAPTER

SEVEN

5. STATICALLY DETERMINATE BEAM

This example has been taken from Ex 12.14 in the Hibbeler textbook [2].

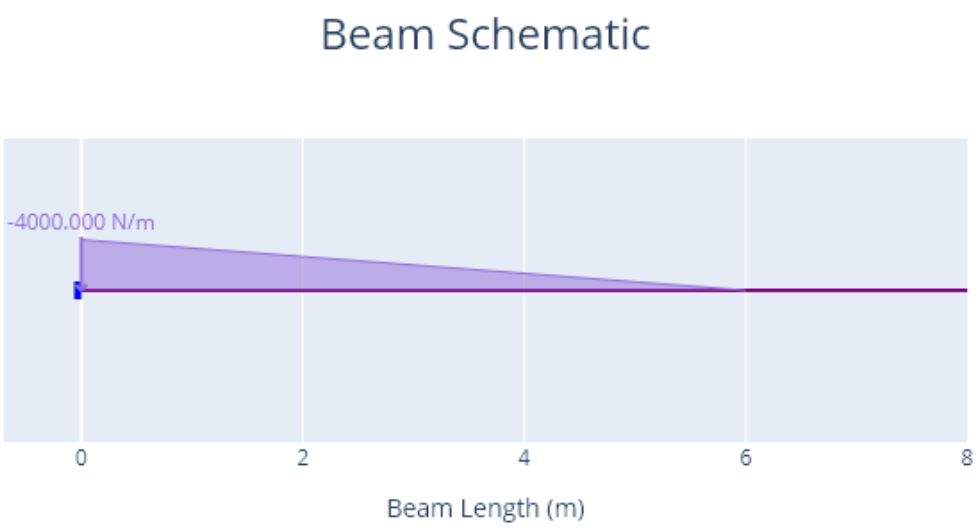
7.1 Specifications

An 8 m long cantilever AB is fixed at A (x = 0 m).

The beam is subject to a trapezoidal load of magnitude 4000 N/m acting downwards at x = 0 m, linearly decreasing in magnitude to a value of 0 N/m at x = 6 m.

Determine the displacement at B (x = 8 m) as a function of EI.

A diagram of the problem is shown below.



7.2 Results

The deflection is -244800.0036000013 (N.m3) / EI (N.m2)

7.3 Code

```
# Statically Determinate beam (Ex 12.14 Hibbeler)
# Determine the displacement at x = 8m for the following structure
# 8 m long fixed at A (x = 0m)
# A trapezoidal load of - 4000 N/m at x = 0 m descending to 0 N/m at x = 6 m.

beam = Beam(8, E=1, I = 1)      ##EI Defined to be 1 get the deflection as a function of␣
↪EI

a = Support(0, (1,1,1))              ##explicitly stated although this is equivalent to␣
↪Support() as the defaults are for a cantilever on the left of the beam.

load_1 = TrapezoidalLoadV((-4000,0),(0,6))

beam.add_supports(a)
beam.add_loads(load_1)

beam.analyse()
print(f"Deflection is {beam.get_deflection(8)} N.m3 / EI (N.mm2)")

fig = beam.plot_beam_internal()
fig.show()
# Note: all plots are correct, deflection graph shape is correct but for actual␣
↪deflection values will need real EI properties.

##save the results as a pdf (optional)
# Can save figure using `fig.write_image("./results.pdf")` (can change extension to be
# png, jpg, svg or other formats as reired). Requires pip install -U kaleido
```

CHAPTER

EIGHT

6. STATICALLY INDETERMINATE BEAM

This example has been taken from Ex 12.21 in the Hibbeler textbook [2].

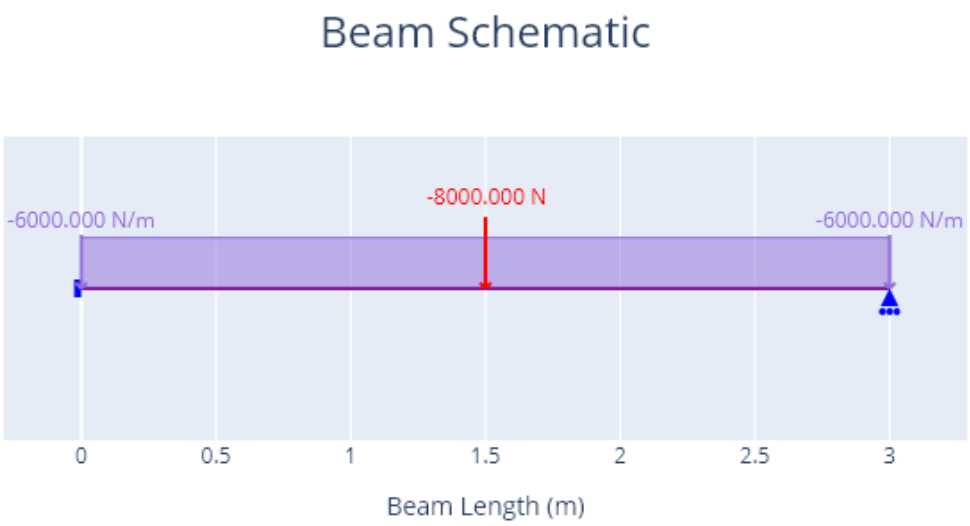
8.1 Specifications

A 3 m long propped cantilever AB is fixed at A (x = 0 m), and supported on a roller at B (x = 3 m).

The beam is subject to a load of 8000 N acting downwards at the midspan, and a UDL of 6000 N/m across the length of the support.

E and I are constant.

A diagram of the problem is shown below.

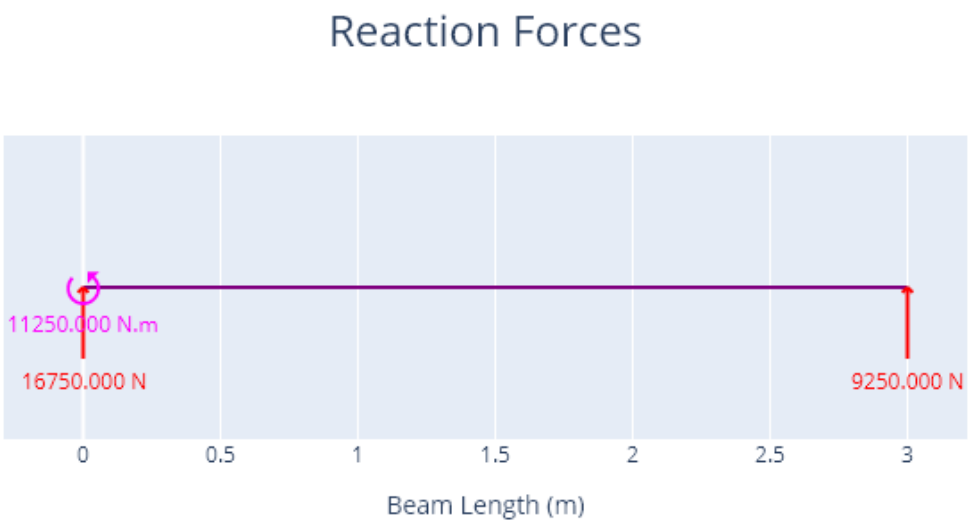


8.2 Results

The following values can be directly extracted using the *get_shear_force*, *get_bending_moment* and *get_reaction* methods:

1. The absolute maximum shear force –> 16750 N
2. The absolute maximum bending moment –> 11250 N.m
3. The reaction at B –> 9250 N

A plot of the reactions is shown below.

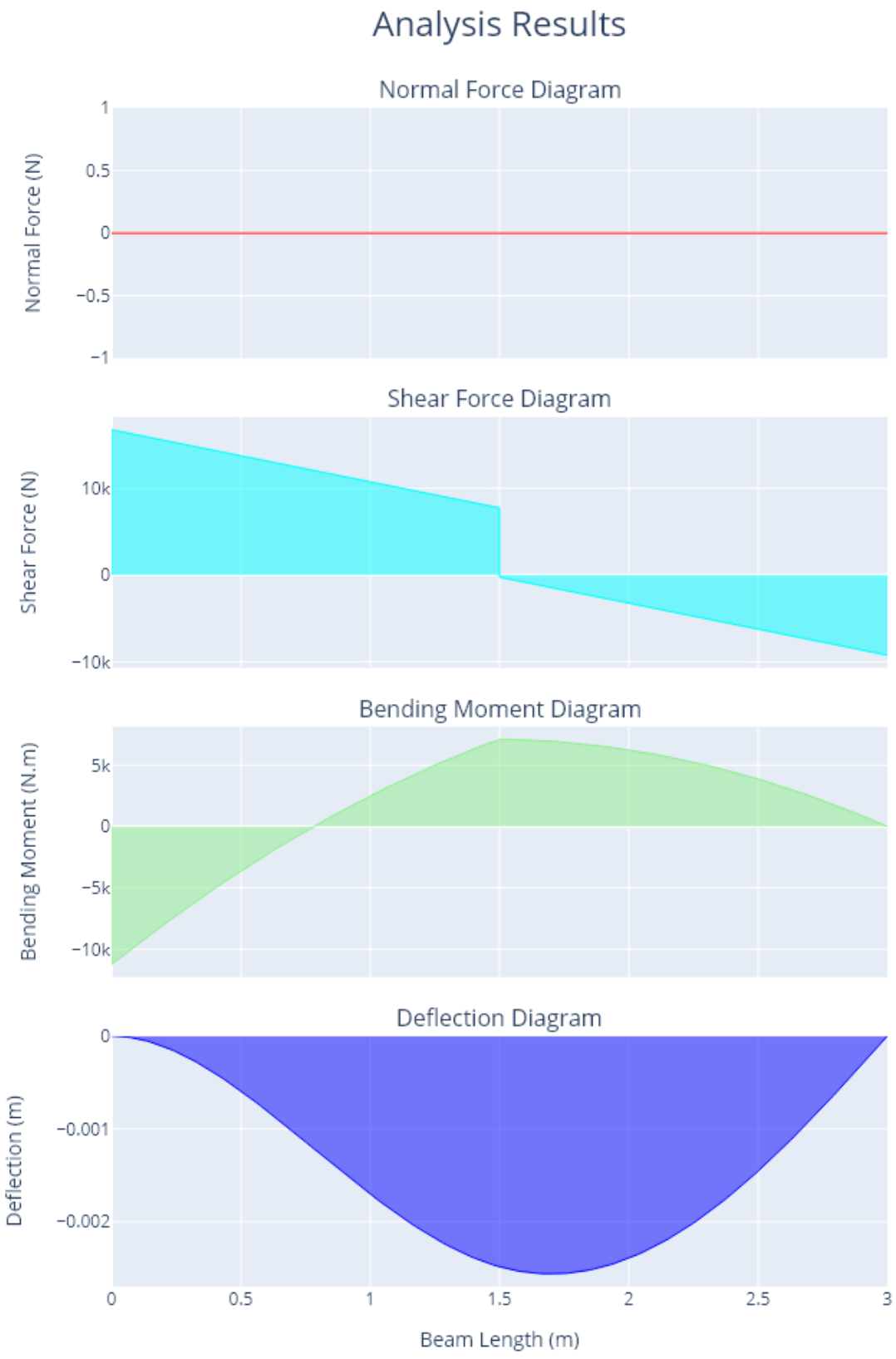


A plot of the axial force, shear force, and bending moments is shown below. A deflection graph is also presented however this depends on the beam properties E and I which werent included in this question. As a default the values E and I are taken as the values for a 150UB18.0 steel beam.

8.3 Code

```
# Statically Indeterminate beam (Ex 12.21 Hibbeler)
# Determine the reactions at the roller support B of the beam described below:
# 3 m long, fixed at A (x = 0 m), roller support at B (x=3 m),
# vertical point load at midpan of 8000 N, UDL of 6000 N/m, EI constant.

beam = Beam(3)

a = Support(0,(1,1,1))
b = Support(3,(0,1,0))

load_1 = PointLoadV(-8000,1.5)
load_2 = UDLV(-6000, (0,3))
```

(continues on next page)


Analysis Results
Normal Force Diagram
Normal Force (N)
1
0.5
0
−0.5
−1
Shear Force Diagram
Shear Force (N)
10k
0
−10k
Bending Moment Diagram
Bending Moment (N.m)
5k
0
−5k
−10k
Deflection Diagram
Deflection (m)
0
−0.001
−0.002
0
0.5
1
1.5
2
2.5
3
Beam Length (m)

(continued from previous page)

```
beam.add_supports(a,b)
beam.add_loads(load_1,load_2)

beam.analyse()

print(f"The beam has an absolute maximum shear force of: {beam.get_shear_force(return_
↪absmax=True)} N")
print(f"The beam has an absolute maximum bending moment of: {beam.get_bending_
↪moment(return_absmax=True)} N.mm")
print(f"The beam has a vertical reaction at B of: {beam.get_reaction(3,'y')} N")

fig1 = beam.plot_beam_external()

fig2 = beam.plot_beam_internal()

fig1.update_layout(width=600)
fig2.update_layout(width=600)

fig_1.write_html("./example_1_external.html")
fig_2.write_html("./example_1_internal.html")
```

CHAPTER

NINE

7. SPRING SUPPORTED BEAM

This example has been taken from Ex 12.16 in the Hibbeler textbook [2].

9.1 Specifications

A 3 m long beam has spring supports at A (x = 0 m) and B (x = 3 m).

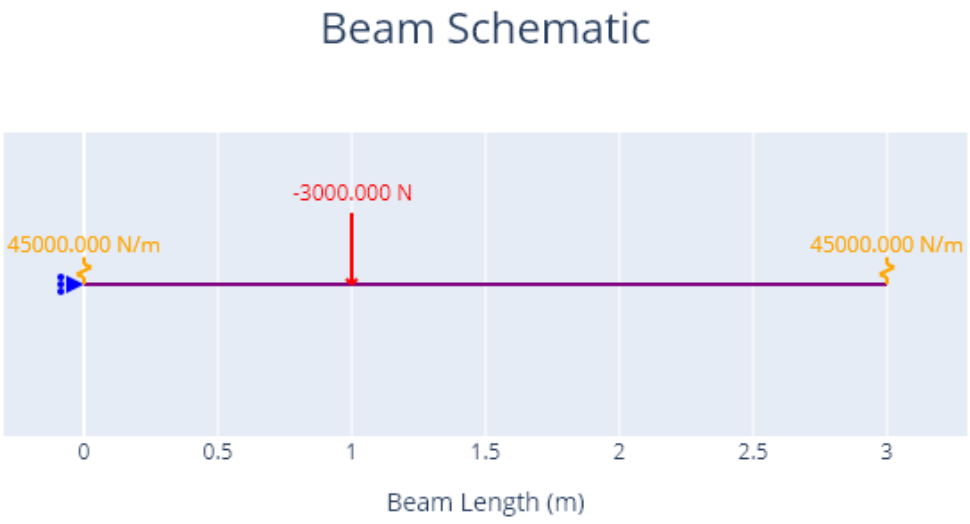
Both spring supports have a stiffness of 45 kN/m.

A downwards vertical point load of magnitude 3000 N acts at x = 1 m.

The beam has a Young's Modulus (E) of 200 GPa and a second moment of area (I) of 4.6875*10**-6 m4.

Determine the vertical displacement at x = 1 m.

A diagram of the problem is shown below.



9.2 Results

The deflection is -0.0370370392 m

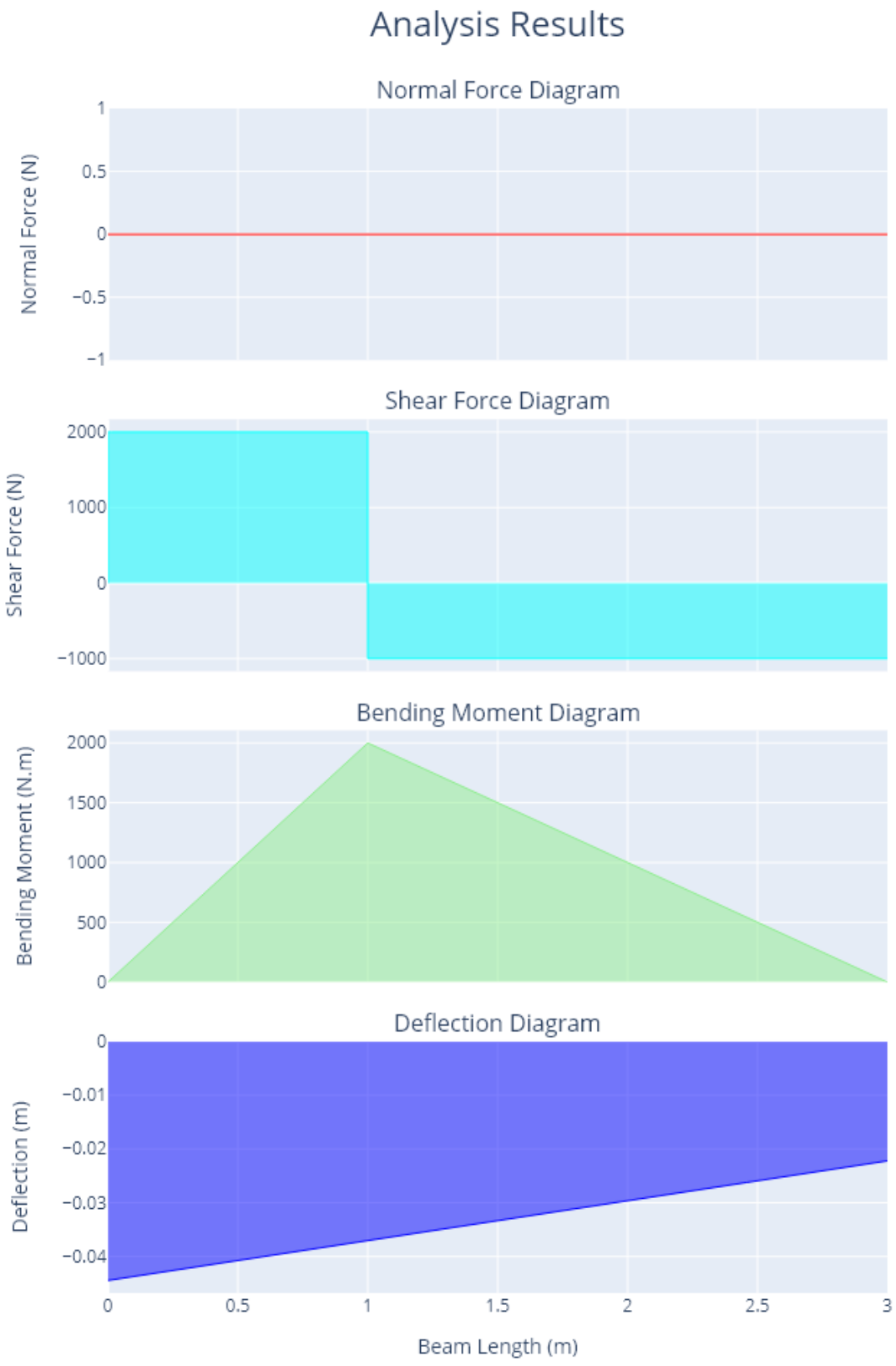
A plot of the reactions is shown below.

A plot of the axial force, shear force, and bending moments is shown below.

9.3 Code

```
# Spring Supported beam (Ex 12.16 Hibbeler)
# Determine the vertical displacement at x = 1 m for the beam detailed below:
# 3 m long, spring of ky = 45 kN/m at A (x = 0 m) and B (x = 3 m), vertical point load␣
↪at x = 1 m of 3000 N, E = 200 GPa, I = 4.6875*10**-6 m4.

# when initializing beam we should specify E and I. Units should be expressed in MPa (N/
↪mm2) for E, and mm4 for I
beam = Beam(3, E=(200)*10**3, I=(4.6875*10**-6)*10**12)

# creating supports, note that an x support must be specified even when there are no x␣
↪forces. This will not affect the accuracy or reliability of results.
# Also note that ky units are kN/m in the problem but must be in N/m for the program to␣
↪work correctly.
a = Support(0, (1,1,0), ky = 45000)
b = Support(3, (0,1,0), ky = 45000)

load_1 = PointLoadV(-3000, 1)

beam.add_supports(a,b)
beam.add_loads(load_1)
```

(continues on next page)


Analysis Results
Normal Force Diagram
Normal Force (N)
1
0.5
0
−0.5
−1
Shear Force Diagram
Shear Force (N)
2000
1000
0
−1000
Bending Moment Diagram
Bending Moment (N.m)
2000
1500
1000
500
0
Deflection Diagram
Deflection (m)
0
−0.01
−0.02
−0.03
−0.04
0
0.5
1
1.5
2
2.5
3
Beam Length (m)

(continued from previous page)

```
beam.analyse()

beam.get_deflection(1)

fig1 = beam.plot_beam_external()
fig1.show()

fig2 = beam.plot_beam_internal()
fig2.show()

##results in 38.46 mm deflection ~= 38.4mm specified in textbook (difference only due to␣
↪their rounding)
##can easily check reliability of answer by looking at deflection at the spring supports.
↪ Should equal F/k.
## ie at support A (x = 0 m), the reaction force is 2kN by equilibrium, so our␣
↪deflection is F/K = 2kn / 45*10-3 kN/mm = 44.4 mm (can be seen in plot)
```

CHAPTER

TEN

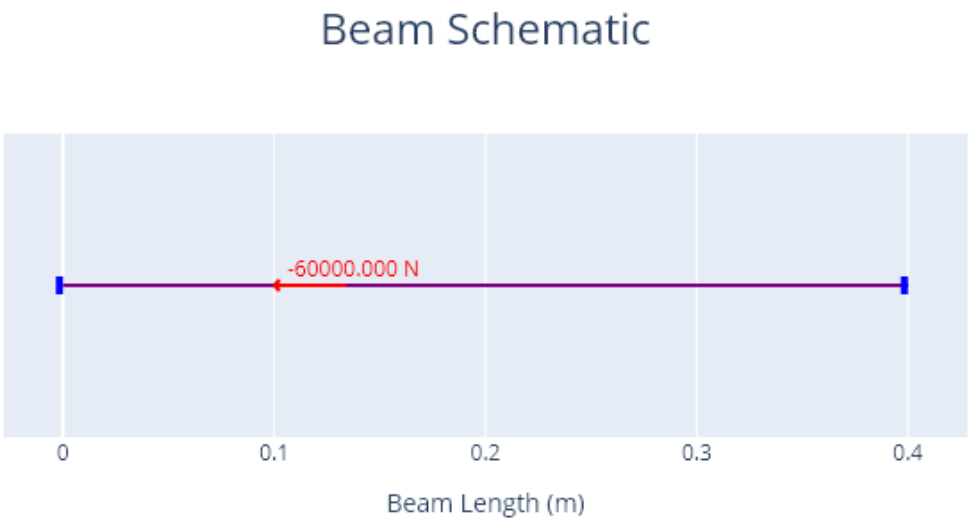
8. AXIALLY LOADED INDETERMINATE BEAM

This example has been taken from Ex 4.13 in the Hibbeler textbook [2].

10.1 Specifications

A rod with constant EA has a force of 60kN applied at x = 0.1 m, and the beam has fixed supports at x=0, and x =0.4 m. Determine the reaction forces.

A diagram of the problem is shown below.



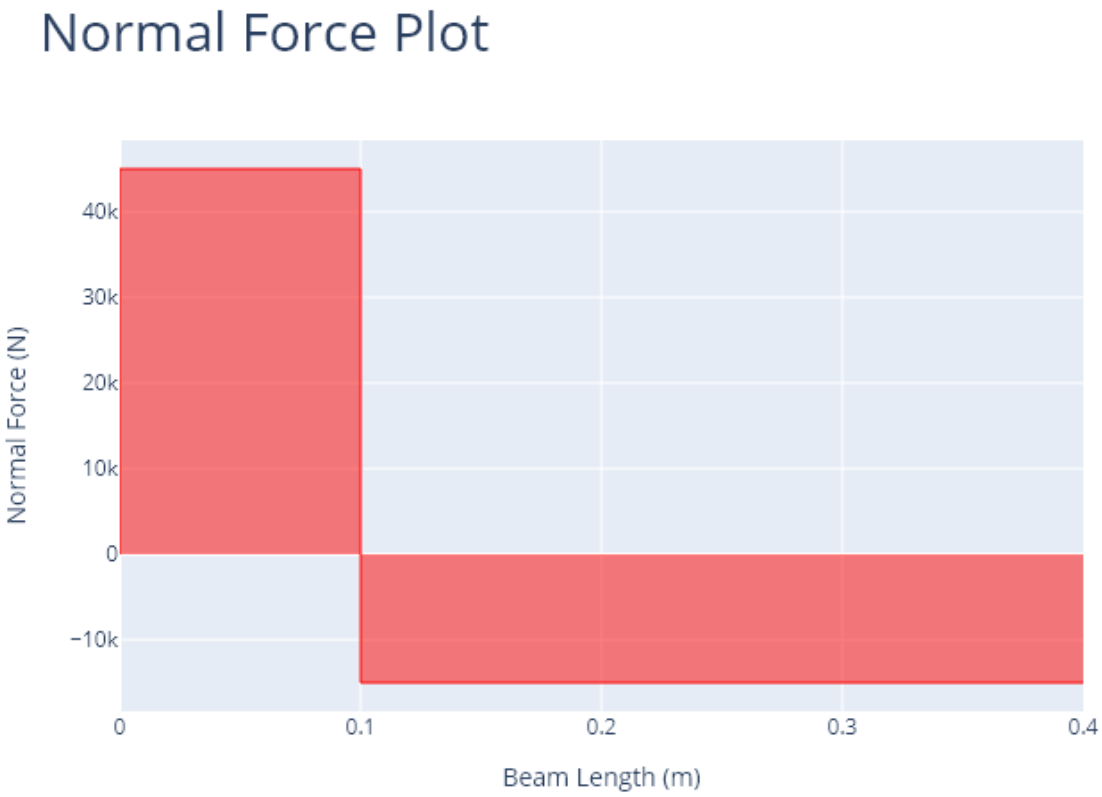
10.2 Results

A plot of the reactions is shown below.

The normal force diagram is presented below.

10.3 Code

```
##AXIAL LOADED INDETERMINATE BEAM (Ex 4.13 Hibbeler)
## A rod with constant EA has a force of 60kN applied at x = 0.1 m, and the beam has␣
↪fixed supports at x=0, and x =0.4 m. Determine the reaction forces.

beam = Beam(0.4)

a = Support()
b = Support(0.4)

load_1 = PointLoadH(-60000, 0.1)

beam.add_supports(a,b)
beam.add_loads(load_1)

beam.analyse()
beam.plot_normal_force()
```


Normal Force Plot
Normal Force (N)
40k
30k
20k
10k
0
−10k
0
0.1
0.2
0.3
0.4
Beam Length (m)

CHAPTER

ELEVEN

INDETERMINATEBEAM REFERENCE

Main module that contains the main class for Beam and auxillary class for Support.

Example

```
>>> beam = Beam(6)
>>> a = Support()
>>> c = Support(6,(0,1,0))
>>> beam.add_supports(a,c)
>>> beam.add_loads(PointLoadV(-1500,3))
>>> beam.analyse()
>>> beam.plot_beam_external()
>>> beam.plot_beam_internal()
```

11.1 Support

class indeterminatebeam.**Support**(*coord=0*, *fixed=(1, 1, 1)*, *kx=None*, *ky=None*)

A class to represent a support.

11.1.1 Attributes:

_position: float
x coordinate of support on a beam (default 0)

_stiffness: tuple of 3 floats or infinity
stiffness K (default units N/m) for movement in x, y and bending. oo represents infinity in sympy and means a completely fixed conventional support, and 0 means free to move.

_DOF
[tuple of 3 booleans] Degrees of freedom that are restraint on a beam for movement in x, y and bending. 1 represents that a reaction force exists and 0 represents free (default (1,1,1))

_fixed: tuple of 3 booleans
Degrees of freedom that are completely fixed on a beam for movement in x, y and bending. 1 represents fixed and 0 represents free or spring (default (1,1,1))

Examples

```
>>> # Creates a fixed suppot at location 0
>>> Support(0, (1,1,1))
>>> # Creates a pinned support at location 5 m
>>> Support(5, (1,1,0))
>>> # Creates a roller support at location 5.54 m
>>> Support(5.54, (0,1,0))
>>> # Creates a y direction spring support at location 7.5 m
>>> Support(7.5, (0,1,0), ky = 5)
```

indeterminatebeam.Support.**__init__**(*self*, *coord=0*, *fixed=(1, 1, 1)*, *kx=None*, *ky=None*)

Constructs all the necessary attributes for the Support object.

11.1.2 Parameters:

coord: float
x coordinate of support on a beam (default unit m) (default 0)

fixed: tuple of 3 booleans
Degrees of freedom that are fixed on a beam for movement in x, y and bending. 1 represents fixed and 0 represents free (default (1,1,1))

kx :
stiffness of x support (default unit N/m), if set will overide the value placed in the fixed tuple. (default = None)

ky
[(positive number)] stiffness of y support (default unit N/m), if set will overide the value placed in the fixed tuple. (default = None)

11.2 Beam

class indeterminatebeam.**Beam**(*span: float = 5*, *E=200000000000*, *I=9.05e-06*, *A=0.23*)

Represents a one-dimensional beam that can take axial and tangential loads.

_x0
Left end coordinate of beam (always defined as 0).

Type
float

_x1
Right end coordinate of beam (default unit m).

Type
float

_E
Young's Modulus of the beam (default unit N/m2 or Pa)

Type
float

_I

Second Moment of Area of the beam (default unit m4)

Type
float

_A

Cross-sectional area of the beam (default unit m2)

Type
float

_loads

list of load objects associated with the beam

Type
list

_supports

A list of support objects associated with the beam.

Type
list

_query

A list containing x coordinates that are to have values explicitly written on graphs.

Type
list

_normal_forces

A sympy function representing the internal axial force (default unit N) as a function of x (m).

Type
sympy piecewise function

_shear_forces

A sympy function representing the internal shear force (default unit N) as a function of x (m).

Type
sympy piecewise function

_bending_moments

A sympy function representing the internal bending moments (default unit N.m) as a function of x (m).

Type
sympy piecewise function

_deflection_equation

A sympy function representing the tangential deflection (default unit m) as a function of x (default unit m).

Type
sympy piecewise function

_reactions

A dictionary with keys for support positions. Each key is associated with a list of forces of the form ['x','y','m']

Type
dictionary of lists

_DATA_POINTS

Number of data points generated for plotting (default 200).

Type
integer,

Notes

- Default units are SI units all using N and m, this can however be changed using the update_units() method
- The default units for length, force and bending moment (torque) are in N and m (m, N, N·m)
- The default units for beam properties (E, I, A) are in N and m (N/m2, m4, m2)
- The default unit for spring support stiffness is N/m

indeterminatebeam.Beam.**__init__**(*self*, *span: float = 5*, *E=200000000000*, *I=9.05e-06*, *A=0.23*)

Initializes a Beam object of a given length.

Parameters

- **span** (*float*) – Length of the beam span (default unit m). Must be positive, and the pinned and rolling supports can only be placed within this span. The default value is 5 m.
- **E** (*float*) – Youngs modulus for the beam (default unit Pa). The default value is 200 GPa, which is the youngs modulus for steel.
- **I** (*float*) – Second moment of area for the beam about the z axis (default unit m4). The default value is 9.05*10**-6 m4.
- **A** (*float*) – Cross-sectional area for the beam about the z axis (default unit m2). The default value is 0.23 m4.

Notes

- Default properties are for a 150UB18.0 steel beam.

indeterminatebeam.Beam.**update_units**(*self*, *key='length'*, *unit='m'*, *reset=False*)

Change units used for inputs and outputs.

Parameters

- **key** (*string, default 'length'*) – Identifying property with unit to be changed. One of the following: 'length', 'force', 'moment', 'distributed', 'stiffness', 'A', 'E', 'I', 'deflection'
- **unit** (*string, default 'm'*) – unit to assign should contain a unit balanced representation consisting of the following units: 'mm', 'cm', 'm', 'N', 'kN', 'Pa', 'kPa', 'MPa', 'in', 'ft', 'lbf', 'kip'. Unit combinations are written in the following formats: 'mm2', 'mm4', 'N/m', 'N.m', 'kip/ft2'.
- **reset** (*boolean, default False*) – If True then all units reset to default SI units.

indeterminatebeam.Beam.**add_loads**(*self*, **loads*)

Associate load objects with the beam object.

Parameters
***loads** (*iterable*) – An iterable containing load objects to be applied to the Beam object. Note that the load application point (or segment) must be within the Beam span.

indeterminatebeam.Beam.**remove_loads**(*self*, **loads*, *remove_all=False*)

Unassociate load objects with the beam object.

Parameters

- ***loads** (*iterable*) – An iterable containing load objects to be removed from the Beam object. If load not on beam then does nothing.
- **remove_all** (*boolean*) – If true all loads associated with beam will be removed, by default False.

indeterminatebeam.Beam.**add_supports**(*self*, **supports*)

Associate support objects with the beam object.

Parameters

***supports** (*iterable*) – An iterable containing Support objects to be applied to the Beam object. Note that the load application point (or segment) must be within the Beam span.

indeterminatebeam.Beam.**remove_supports**(*self*, **supports*, *remove_all=False*)

Unassociate support objects with the beam object.

Parameters

- ***supports** (*iterable*) – An iterable containing Support objects to be removed from the Beam object. If support not on beam then does nothing.
- **remove_all** (*boolean*) – If true all supports associated with beam will be removed, by default False.

indeterminatebeam.Beam.**analyse**(*self*)

Solve the beam structure for reaction and internal forces

indeterminatebeam.Beam.**get_reaction**(*self*, *x_coord*, *direction=None*)

Find the reactions of a support at position x.

Parameters

- **x_coord** (*float*) – The x_coordinates on the beam to be substituted into the equation. List returned (if bools all false)
- **direction** (*str ('x','y' or 'm')*) – The direction of the reaction force to be returned. If not specified all are returned in a list.

Returns

- *int* – If direction is 'x', 'y', or 'm' will return an integer representing the reaction force of the support in that direction at location x_coord.
- *list of ints* – If direction = None, will return a list of 3 integers, representing the reaction forces of the support ['x','y','m'] at location x_coord.
- *None* – If there is no support at the x coordinate specified.

indeterminatebeam.Beam.**get_bending_moment**(*self*, **x_coord*, *return_max=False*, *return_min=False*, *return_absmax=False*)

Find the bending moment(s) on the beam object.

Parameters

- **x_coord** (*list*) – The x_coordinates on the beam to be substituted into the equation. List returned (if bools all false)
- **return_max** (*bool*) – return max value of function if true

- **return_min** (*bool*) – return minx value of function if true
- **return_absmax** (*bool*) – return absolute max value of function if true

Returns

- *int* – Max, min or absmax value of the bending moment depending on which parameters are set. If single x-coordinate set then also returns int.
- *list of ints* – If x-coordinates are specfied value at x-coordinates for func.

Notes

- Priority of query parameters is return_max, return_min, return_absmax, x_coord (if more than 1 of the parameters are specified).

indeterminatebeam.Beam.**get_shear_force**(*self*, **x_coord*, *return_max=False*, *return_min=False*, *return_absmax=False*)

Find the shear force(s) on the beam object.

Parameters

- **x_coord** (*list*) – The x_coordinates on the beam to be substituted into the equation. List returned (if bools all false)
- **return_max** (*bool*) – return max value of function if true
- **return_min** (*bool*) – return minx value of function if true
- **return_absmax** (*bool*) – return absolute max value of function if true

Returns

- *int* – Max, min or absmax value of the shear force depending on which parameters are set. If single x-coordinate set then also returns int.
- *list of ints* – If x-coordinates are specfied value at x-coordinates for func.

Notes

- Priority of query parameters is return_max, return_min, return_absmax, x_coord (if more than 1 of the parameters are specified).

indeterminatebeam.Beam.**get_normal_force**(*self*, **x_coord*, *return_max=False*, *return_min=False*, *return_absmax=False*)

Find the normal force(s) on the beam object.

Parameters

- **x_coord** (*list*) – The x_coordinates on the beam to be substituted into the equation. List returned (if bools all false)
- **return_max** (*bool*) – return max value of function if true
- **return_min** (*bool*) – return minx value of function if true
- **return_absmax** (*bool*) – return absolute max value of function if true

Returns

- *int* – Max, min or absmax value of the normal force depending on which parameters are set. If single x-coordinate set then also returns int.
- *list of ints* – If x-coordinates are specfied value at x-coordinates for func.

Notes

- Priority of query parameters is return_max, return_min, return_absmax, x_coord (if more than 1 of the parameters are specified).

indeterminatebeam.Beam.**get_deflection**(*self*, **x_coord*, *return_max=False*, *return_min=False*, *return_absmax=False*)

Find the deflection(s) on the beam object.

Parameters

- **x_coord** (*list*) – The x_coordinates on the beam to be substituted into the equation. List returned (if bools all false)
- **return_max** (*bool*) – return max value of function if true
- **return_min** (*bool*) – return minx value of function if true
- **return_absmax** (*bool*) – return absolute max value of function if true

Returns

- *int* – Max, min or absmax value of the deflection depending on which parameters are set. If single x-coordinate set then also returns int.
- *list of ints* – If x-coordinates are specfied value at x-coordinates for func.

Notes

- Priority of query parameters is return_max, return_min, return_absmax, x_coord (if more than 1 of the parameters are specified).

indeterminatebeam.Beam.**add_query_points**(*self*, **x_coords*)

Document the forces on a beam at position x_coord when plotting.

Parameters

x_coord (*list*) – The x_coordinates on the beam to be queried on plot.

indeterminatebeam.Beam.**remove_query_points**(*self*, **x_coords*, *remove_all=False*)

Remove a query point added by add_query_points function.

Parameters

- **x_coord** (*list*) – The x_coordinates on the beam to be removed from query on plot.
- **remove_all** (*boolean*) – If true all query points will be removed.

indeterminatebeam.Beam.**plot_beam_external**(*self*)

Generates a single figure with 2 plots corresponding respectively to:

- a schematic of the loaded beam
- reaction force diagram

Returns
figure – Returns a handle to a figure with the 2 subplots.

Return type
plotly.graph_objs._figure.Figure

indeterminatebeam.Beam.**plot_beam_internal**(*self*, *reverse_x=False*, *reverse_y=False*)

Generates a single figure with 4 plots corresponding respectively to:

- normal force diagram,
- shear force diagram,
- bending moment diagram, and
- deflection diagram

Parameters

- **reverse_x** (*bool, optional*) – reverse the x axes, by default False
- **reverse_y** (*bool, optional*) – reverse the y axes, by default False

Returns
figure – Returns a handle to a figure with the 4 subplots.

Return type
plotly.graph_objs._figure.Figure

indeterminatebeam.Beam.**plot_beam_diagram**(*self*, *fig=None*, *row=None*, *col=None*)

Returns a schematic of the beam and all the loads applied on it

Parameters

- **fig** (*bool, optional*) – Figure to append subplot diagram too. If creating standalone figure then None, by default None
- **row** (*int, optional*) – row number if subplot, by default None
- **col** (*int, optional*) – column number if subplot, by default None

Returns
figure – Returns a handle to a figure with the beam schematic.

Return type
plotly.graph_objs._figure.Figure

indeterminatebeam.Beam.**plot_reaction_force**(*self*, *fig=None*, *row=None*, *col=None*)

Returns a plot of the beam with reaction forces.

Parameters

- **fig** (*bool, optional*) – Figure to append subplot diagram too. If creating standalone figure then None, by default None
- **row** (*int, optional*) – row number if subplot, by default None
- **col** (*int, optional*) – column number if subplot, by default None

Returns
figure – Returns a handle to a figure with reaction forces.

Return type
plotly.graph_objs._figure.Figure

indeterminatebeam.Beam.**plot_normal_force**(*self*, *reverse_x=False*, *reverse_y=False*, *switch_axes=False*, *fig=None*, *row=None*, *col=None*)

Returns a plot of the normal force as a function of the x-coordinate.

Parameters

- **reverse_x** (*bool, optional*) – reverse the x axes, by default False
- **reverse_y** (*bool, optional*) – reverse the y axes, by default False
- **switch_axes** (*bool, optional*) – switch the x and y axis, by default False
- **fig** (*bool, optional*) – Figure to append subplot diagram too. If creating standalone figure then None, by default None
- **row** (*int, optional*) – row number if subplot, by default None
- **col** (*int, optional*) – column number if subplot, by default None

Returns
figure – Returns a handle to a figure with the normal force diagram.

Return type
plotly.graph_objs._figure.Figure

indeterminatebeam.Beam.**plot_shear_force**(*self*, *reverse_x=False*, *reverse_y=False*, *switch_axes=False*, *fig=None*, *row=None*, *col=None*)

Returns a plot of the shear force as a function of the x-coordinate.

Parameters

- **reverse_x** (*bool, optional*) – reverse the x axes, by default False
- **reverse_y** (*bool, optional*) – reverse the y axes, by default False
- **switch_axes** (*bool, optional*) – switch the x and y axis, by default False
- **fig** (*bool, optional*) – Figure to append subplot diagram too. If creating standalone figure then None, by default None
- **row** (*int, optional*) – row number if subplot, by default None
- **col** (*int, optional*) – column number if subplot, by default None

Returns
figure – Returns a handle to a figure with the shear force diagram.

Return type
plotly.graph_objs._figure.Figure

indeterminatebeam.Beam.**plot_bending_moment**(*self*, *reverse_x=False*, *reverse_y=False*, *switch_axes=False*, *fig=None*, *row=None*, *col=None*)

Returns a plot of the bending moment as a function of the x-coordinate.

Parameters

- **reverse_x** (*bool, optional*) – reverse the x axes, by default False
- **reverse_y** (*bool, optional*) – reverse the y axes, by default False
- **switch_axes** (*bool, optional*) – switch the x and y axis, by default False
- **fig** (*bool, optional*) – Figure to append subplot diagram too. If creating standalone figure then None, by default None
- **row** (*int, optional*) – row number if subplot, by default None

- **col** (*int, optional*) – column number if subplot, by default None

Returns
figure – Returns a handle to a figure with the bending moment diagram.

Return type
plotly.graph_objs._figure.Figure

indeterminatebeam.Beam.**plot_deflection**(*self*, *reverse_x=False*, *reverse_y=False*, *switch_axes=False*, *fig=None*, *row=None*, *col=None*)

Returns a plot of the beam deflection as a function of the x-coordinate.

Parameters

- **reverse_x** (*bool, optional*) – reverse the x axes, by default False
- **reverse_y** (*bool, optional*) – reverse the y axes, by default False
- **switch_axes** (*bool, optional*) – switch the x and y axis, by default False
- **fig** (*bool, optional*) – Figure to append subplot diagram too. If creating standalone figure then None, by default None
- **row** (*int, optional*) – row number if subplot, by default None
- **col** (*int, optional*) – column number if subplot, by default None

Returns
figure – Returns a handle to a figure with the deflection diagram.

Return type
plotly.graph_objs._figure.Figure

11.3 PointLoads

class indeterminatebeam.**PointTorque**(*force=0*, *coord=0*)

Point clockwise torque.

11.3.1 Parameters:

force: float
Torque load (default units N.m), (named force for consistency with other load types)

coord: float
x coordinate of torque on beam (default units m)

Examples

```
>>> # 30 N·m (anti-clockwise) torque at x = 4 m
>>> motor_torque = PointTorque(30, 4)
```

Note: Anti-clockwise is positive

class indeterminatebeam.**PointLoad**(*force=0*, *coord=0*, *angle=0*)

Point load.

11.3.2 Parameters:

Force: float
Force load (default units m)

coord: float
x coordinate of load on beam (default units m)

angle: float
angle of point load where: - 0 degrees is purely horizontal +ve - 90 degrees is purely vertical +ve - 180 degrees is purely horizontal -ve of force sign specified.

Examples

```
>>> # 100 N towards the right at x = 9 m
>>> external_force = PointLoad(100, 9, 90)
>>> # 300 N downwards at x = 3 m
>>> external_force = PointLoad(-300, 3, 0)
>>> external_force
    PointLoad(force=-300, coord=3, angle=0)
```

class indeterminatebeam.**PointLoadV**(*force=0*, *coord=0*)

Vertical Point Load.

11.3.3 Parameters:

Force: float
Force load (default units N)

coord: float
x coordinate of load on beam (default units m)

Examples

```
>>> # 100 N towards the right at x = 9 m
>>> external_force = PointLoad(100, 9)
```

Note: Positive force acts up.

class indeterminatebeam.**PointLoadH**(*force=0*, *coord=0*)

Horizontal Point Load.

11.3.4 Parameters:

Force: float
Force load (default units m)

coord: float
x coordinate of load on beam (default units m)

Examples

```
>>> # 100 N up at x = 9 m
>>> external_force = PointLoad(100, 9)
```

Note: Positive force acts right.

11.4 DistributedLoads

class indeterminatebeam.**UDL**(*force=0*, *span=(0, 0)*, *angle=0*)

Uniformly Distributed Load (UDL).

Parameters

- **force** (*int, optional*) – UDL load (default units N/m), by default 0
- **span** (*tuple of floats*) – A tuple containing the starting and ending coordinate that the UDL is applied to (default units m).
- **angle** (*float*) – angle of point load where: - 0 degrees is purely horizontal +ve - 90 degrees is purely vertical +ve - 180 degrees is purely horizontal -ve of force sign specified.

Examples

```
>>> # load of 1000 N/m from 1 m <= x <= 4 m (vertical)
>>> self_weight = UDL(1000, (1, 4), 90)
```

class indeterminatebeam.**UDLV**(*force=0*, *span=(0, 0)*)

Vertical Uniformly Distributed Load.

Parameters

- **Force** (*float*) – Force load (default units N/m)
- **span** (*tuple of floats*) – A tuple containing the starting and ending coordinate that the UDL is applied to (default units m).

Examples

```
>>> # load of 1000 N/m (acting down) from 1 <= x <= 4 m
>>> self_weight = UDL(-1000, (1, 4))
```

Note: Positive force acts up.

class indeterminatebeam.**UDLH**(*force=0*, *span=(0, 0)*)

Horizontal Uniformly Distributed Load.

Parameters

- **Force** (*float*) – Force load (default units N/m)
- **span** (*tuple of floats*) – A tuple containing the starting and ending coordinate that the UDL is applied to (default units m).

Examples

```
>>> # load of 1000 N/m (acting right) from 1 <= x <= 4 m
>>> self_weight = UDL(-1000, (1, 4))
```

Note: Positive force acts right.

class indeterminatebeam.**TrapezoidalLoad**(*force=(0, 0)*, *span=(0, 0)*, *angle=0*)

Trapezoidal Distributed Load.

Parameters

- **force** (*tuple of floats*) – A tuple containing the starting and ending loads of the trapezoidal load (default units N/m).
- **span** (*tuple of floats*) – A tuple containing the starting and ending coordinate that the trapezoidal load is applied to (default units m).
- **angle** (*float*) – angle of point load where: - 0 degrees is purely horizontal +ve - 90 degrees is purely vertical +ve - 180 degrees is purely horizontal -ve of force sign specified.

Examples

```
>>> # trapezoidal load starting at 2000 N/m at 1 m and ending at 3000 N/m
>>> # at 4 m (vertical)
>>> self_weight = UDL((2000, 3000), (1, 4), 90)
```

class indeterminatebeam.**TrapezoidalLoadV**(*force=(0, 0)*, *span=(0, 0)*)

Vertical Trapezoidal Distributed Load.

Parameters

- **force** (*tuple of floats*) – A tuple containing the starting and ending loads of the trapezoidal load (default units N/m).
- **span** (*tuple of floats*) – A tuple containing the starting and ending coordinate that the trapezoidal load is applied to (default units m).

Examples

```
>>> # trapezoidal load starting at 2000 N/m at 1 m and ending at 3000 N/m
>>> # at 4 m (acting down)
>>> self_weight = UDL((-2000,-3000), (1, 4))
```

Note: Positive force acts up.

class indeterminatebeam.**TrapezoidalLoadH**(*force=(0, 0)*, *span=(0, 0)*)

Horizontal Trapezoidal Distributed Load.

Parameters

- **force** (*tuple of floats*) – A tuple containing the starting and ending loads of the trapezoidal load (default units N/m).
- **span** (*tuple of floats*) – A tuple containing the starting and ending coordinate that the trapezoidal load is applied to (default units m).

Examples

```
>>> # trapezoidal load starting at 2000 N/m at 1 m and ending at 3 N/m
>>> # at 4 m (acting right)
>>> self_weight = UDL((2000, 3000), (1, 4))
```

Note: Positive force acts right.

class indeterminatebeam.**DistributedLoad**(*expr*, *span=(0, 0)*, *angle=0*)

Distributed load, described by its functional form, application interval and the angle of the load relative to the beam.

11.4.1 Parameters:

expr: sympy expression
Sympy expression of the distributed load function expressed using variable x which represents the beam x-coordinate. Requires quotation marks around expression (default units in N/m).

span: tuple of floats

A tuple containing the starting and ending coordinate that
the function is applied to (default units m).

angle: float
angle of point load where: - 0 degrees is purely horizontal +ve - 90 degrees is purely vertical +ve - 180 degrees is purely horizontal -ve of force sign specified.

Examples

```
>>> # Linearly growing load for 0 < x < 2 m
>>> snow_load = DistributedLoad("10 * x + 5", (0, 2), 90)
```

class indeterminatebeam.**DistributedLoadV**(*expr=0*, *span=(0, 0)*)

Vertical distributed load, described by its functional form, application interval and the angle of the load relative to the beam.

11.4.2 Parameters:

expr: sympy expression
Sympy expression of the distributed load function expressed using variable x which represents the beam x-coordinate (default units N/m).Requires quotation marks around expression.

span: tuple of floats

A tuple containing the starting and ending coordinate that
the function is applied to (default units m).

Examples

```
>>> # Linearly growing load (acting down) for 0 < x < 2 m
>>> snow_load = DistributedLoad("-10*x-5", (0, 2))
```

Note: Positive force acts up.

class indeterminatebeam.**DistributedLoadH**(*expr=0*, *span=(0, 0)*)

Horizontal distributed load, described by its functional form, application interval and the angle of the load relative to the beam.

11.4.3 Parameters:

expr: sympy expression
Sympy expression of the distributed load function expressed using variable x which represents the beam x-coordinate (default units N/m). Requires quotation marks around expression.

span: tuple of floats

A tuple containing the starting and ending coordinate that
the function is applied to (default units m).

Examples

```
>>> # Linearly growing load (acting right) for 0 < x < 2 m
>>> snow_load = DistributedLoad("10*x+5", (0, 2))
```

Note: Positive force acts right.

CHAPTER

TWELVE

REFERENCES

A special thanks to Alredo Carella, I have learnt a lot about documenting python code by observing his code in beam-bending.

12.1 Bibliography

BIBLIOGRAPHY

[1] Alfredo Carella. Beambending: a teaching aid for 1-d shear force and bending moment diagrams. *Journal of Open Source Education*, 2(16):65, 2019. URL: https://doi.org/10.21105/jose.00065, doi:10.21105/jose.00065.

[2] Russell Hibbeler. *Mechanics of Materials*. P.Ed Australia, Melbourne, 2013. ISBN 9810694369.

PYTHON MODULE INDEX

INDEX

Symbols

A

B

D

G

I

M

P

R

S

T

U