MODAL ANALYSIS

OpenSeesPy Tutorial 1

TUTORIAL 1. INTRODUCTION

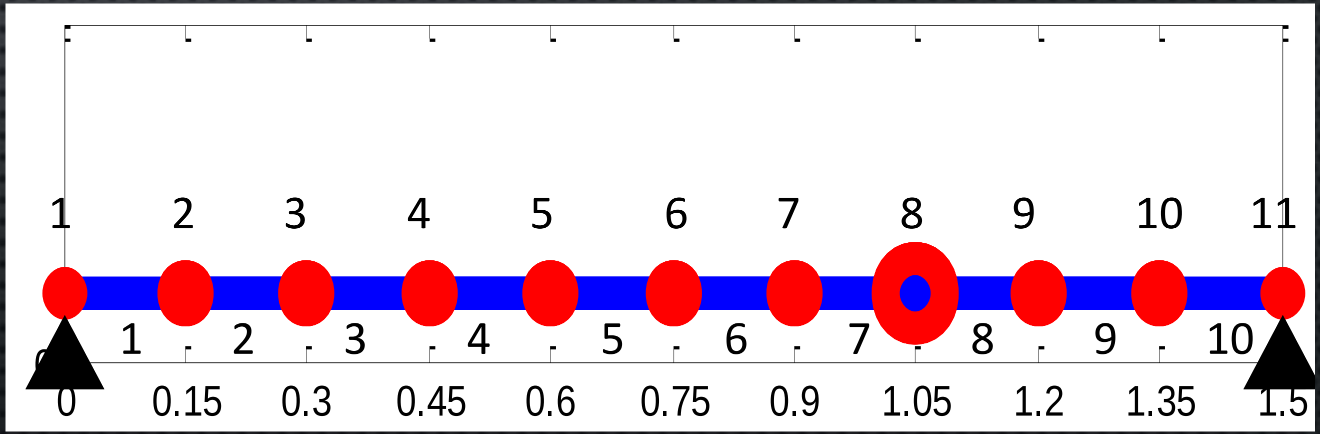
The intention of this tutorial is to guide you through simple models using the OpenSeesPy library. Our primary goal is to provide you with a clear understanding of the fundamental framework of the library and make this document a valuable reference that helps in your learning journey.

In this tutorial, we will focus on conducting modal analysis using OPenSeesPy. His serves as a foundational step in your journey with OpenSeesPy.

The example model used in the tutorial is simple enough to grasp fundamental concepts in structural analysis, but it also provides the opportunity to dive deeper into more complex analyses in subsequent tutorials.

CASE OF STUDY: DYNAMIC ANALYSIS OF A SHAFT WITH MASS IMBALANCE

The figure shows a model of a rotodynamic device that comprises of a 1.5 meter long steel shaft with a diameter of ½ inch. The shaft also features a 40 kg mass disk attached to it. The system is supported at its ends by infinitely rigid simple supports.



CASE OF STUDY: DYNAMIC ANALYSIS OF A SHAFT WITH MASS IMBALANCE (CONTINUED)

Length (L): 1.5 m

Diameter (D): 0.5 inches

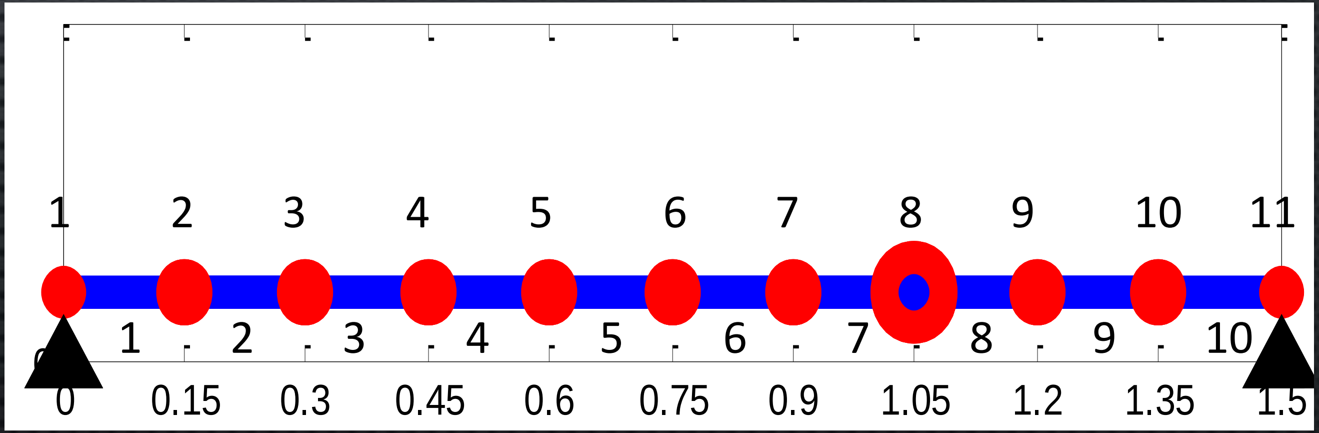
Mass (m): 40 kg

Density (rho): 7850 kg/m^3

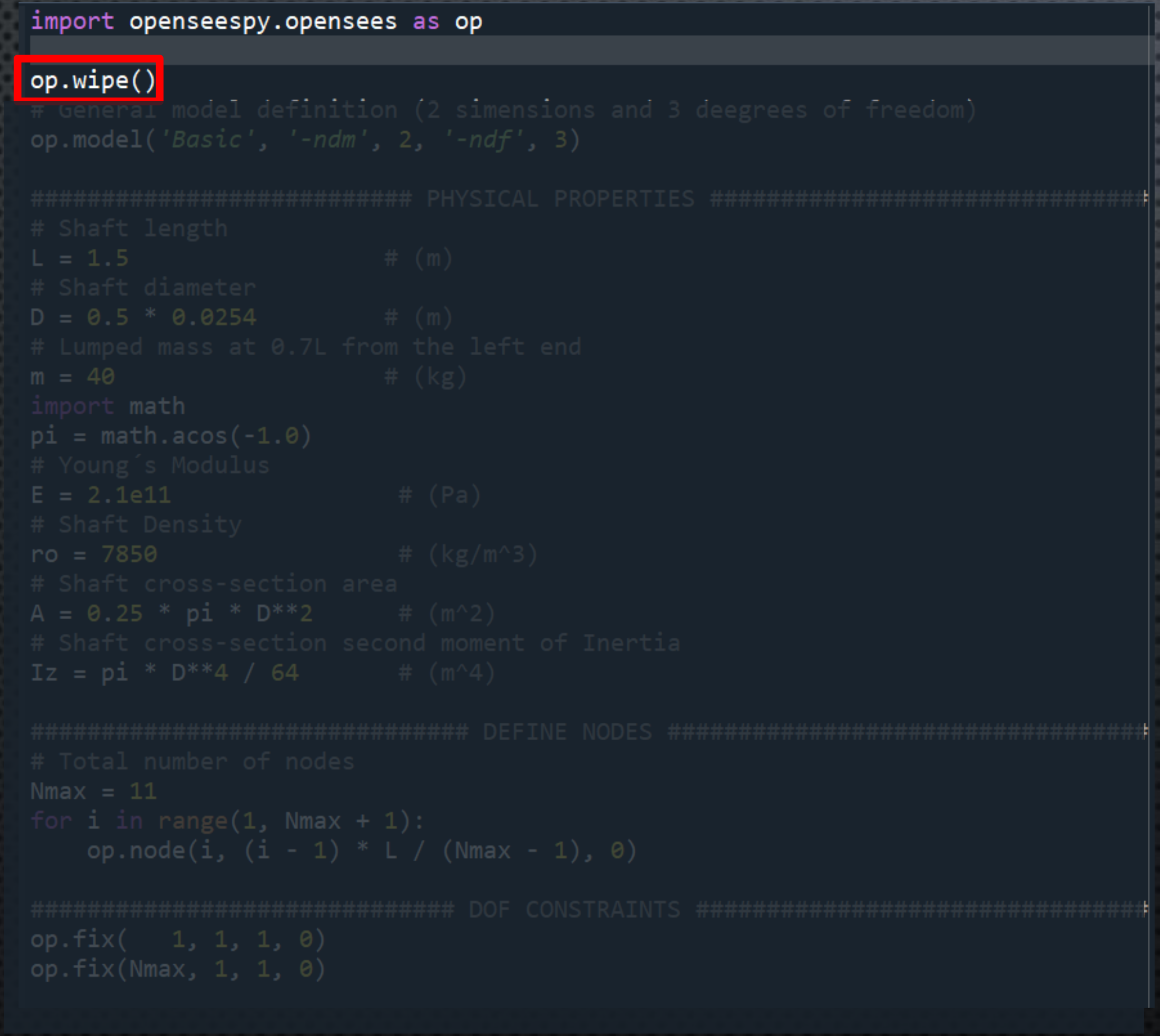
Young's Modulus (E): 2.1 GPa

Simply supported shaft.

Mass excess at 0.7L from left-end.



WIPE COMMAND

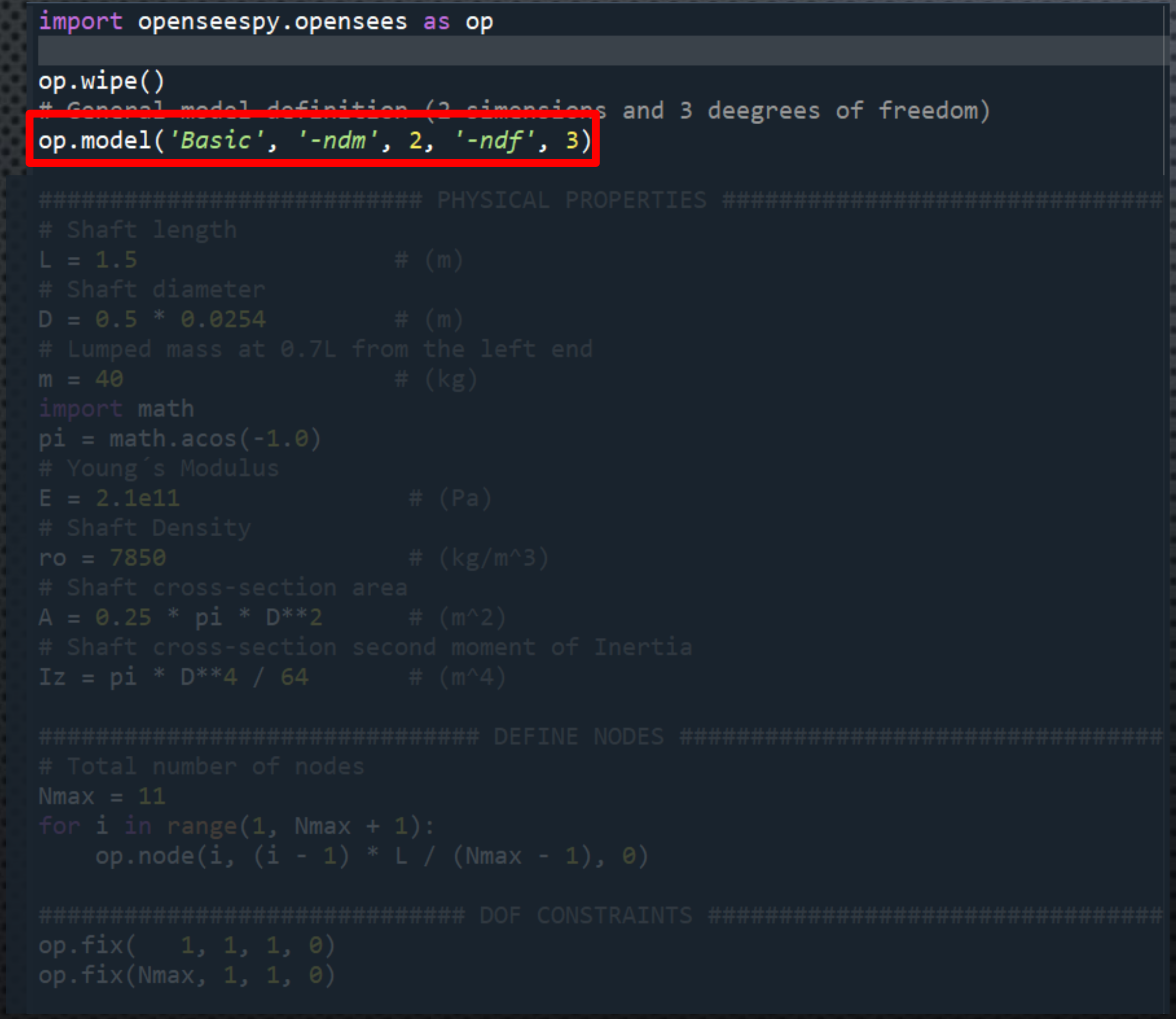


```
import openseespy.opensees as op

op.wipe()
# General model definition (2 simensions and 3 deegrees of freedom)
op.model('Basic', '-ndm', 2, '-ndf', 3)

########################### PHYSICAL PROPERTIES ###############################
# Shaft length
L = 1.5                   # (m)
# Shaft diameter
D = 0.5 * 0.0254          # (m)
# Lumped mass at 0.7L from the left end
m = 40                    # (kg)
import math
pi = math.acos(-1.0)
# Young´s Modulus
E = 2.1e11                 # (Pa)
# Shaft Density
ro = 7850                  # (kg/m^3)
# Shaft cross-section area
A = 0.25 * pi * D**2       # (m^2)
# Shaft cross-section second moment of Inertia
Iz = pi * D**4 / 64        # (m^4)

############################### DEFINE NODES ##################################
# Total number of nodes
Nmax = 11
for i in range(1, Nmax + 1):
    op.node(i, (i - 1) * L / (Nmax - 1), 0)

############################## DOF CONSTRAINTS ################################
op.fix(   1, 1, 1, 0)
op.fix(Nmax, 1, 1, 0)
```

The wipe command ensures that **all OpenSeesPy variables** and previously defined commands are no longer active in the current model.

As a common practice to prevent interference from other analyses, it is recommended to initiate the model with this command.

MODEL DEFINITION



```
import openseespy.opensees as op

op.wipe()
# General model definition (2 dimensions and 3 deegrees of freedom)
op.model('Basic', '-ndm', 2, '-ndf', 3)

############################ PHYSICAL PROPERTIES ##################################
# Shaft length
L = 1.5                         # (m)
# Shaft diameter
D = 0.5 * 0.0254                # (m)
# Lumped mass at 0.7L from the left end
m = 40                          # (kg)
import math
pi = math.acos(-1.0)
# Young´s Modulus
E = 2.1e11                       # (Pa)
# Shaft Density
ro = 7850                        # (kg/m^3)
# Shaft cross-section area
A = 0.25 * pi * D**2             # (m^2)
# Shaft cross-section second moment of Inertia
Iz = pi * D**4 / 64              # (m^4)

############################### DEFINE NODES ###################################
# Total number of nodes
Nmax = 11
for i in range(1, Nmax + 1):
    op.node(i, (i - 1) * L / (Nmax - 1), 0)

############################## DOF CONSTRAINTS #################################
op.fix(   1, 1, 1, 0)
op.fix(Nmax, 1, 1, 0)
```

The model command is a necessary step to define the model domain. The syntax for this command is as follows:

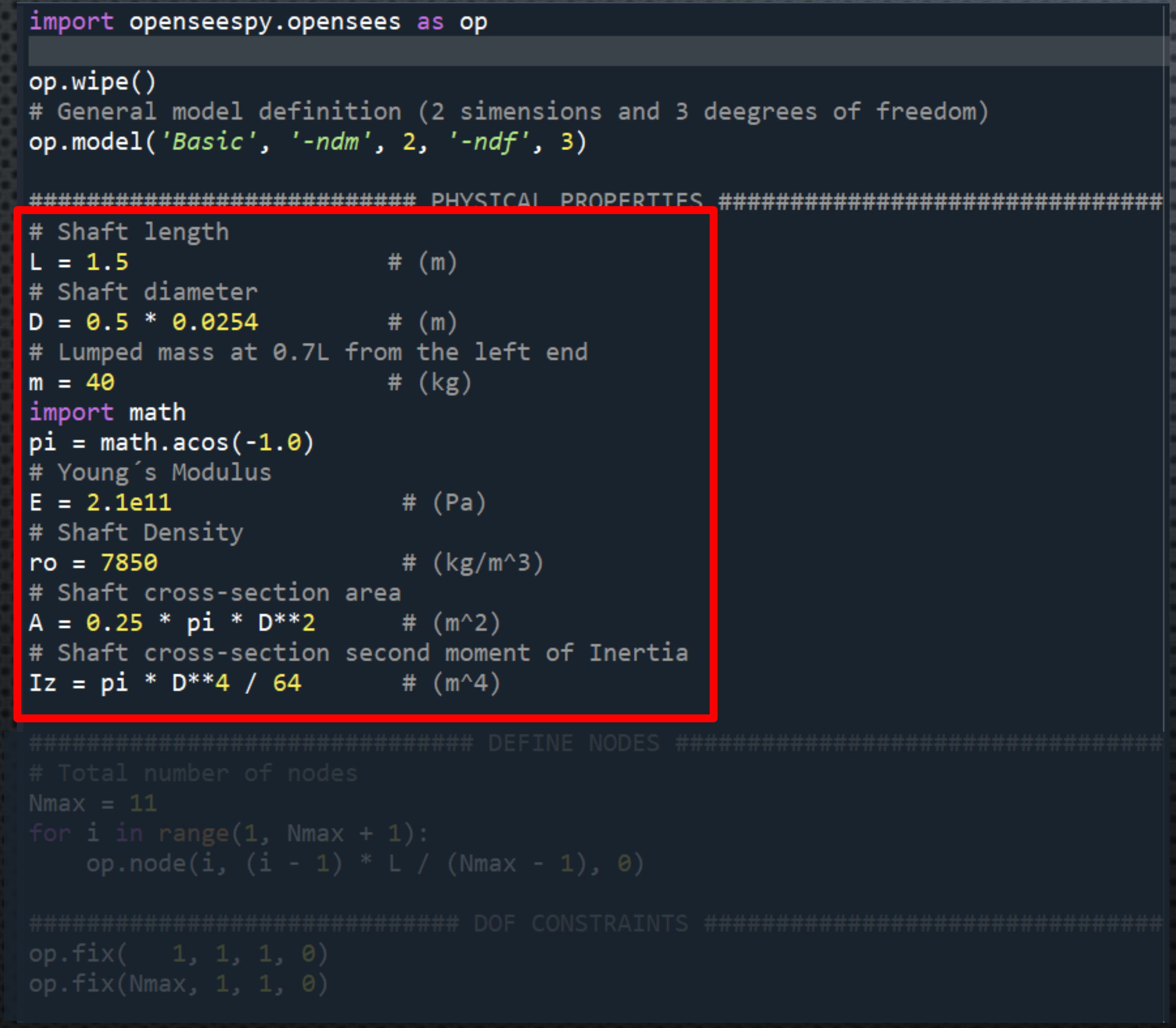
model('basic', '-ndm', ndm, '-ndf', ndf)

Where:

- 'basic' represents the model type.
- '-ndm' denotes the number of dimensions.
- ndm is an integer representing the dimension.
- '-ndf' indicates the number of degrees of freedom.
- ndf is the integer representing the number of degrees of freedom.

It is important to note that ndf is an optional argument, if not specified, its default is the same as ndm.

GEOMETRY VARIABLES AND MATERIAL ATTRIBUTES

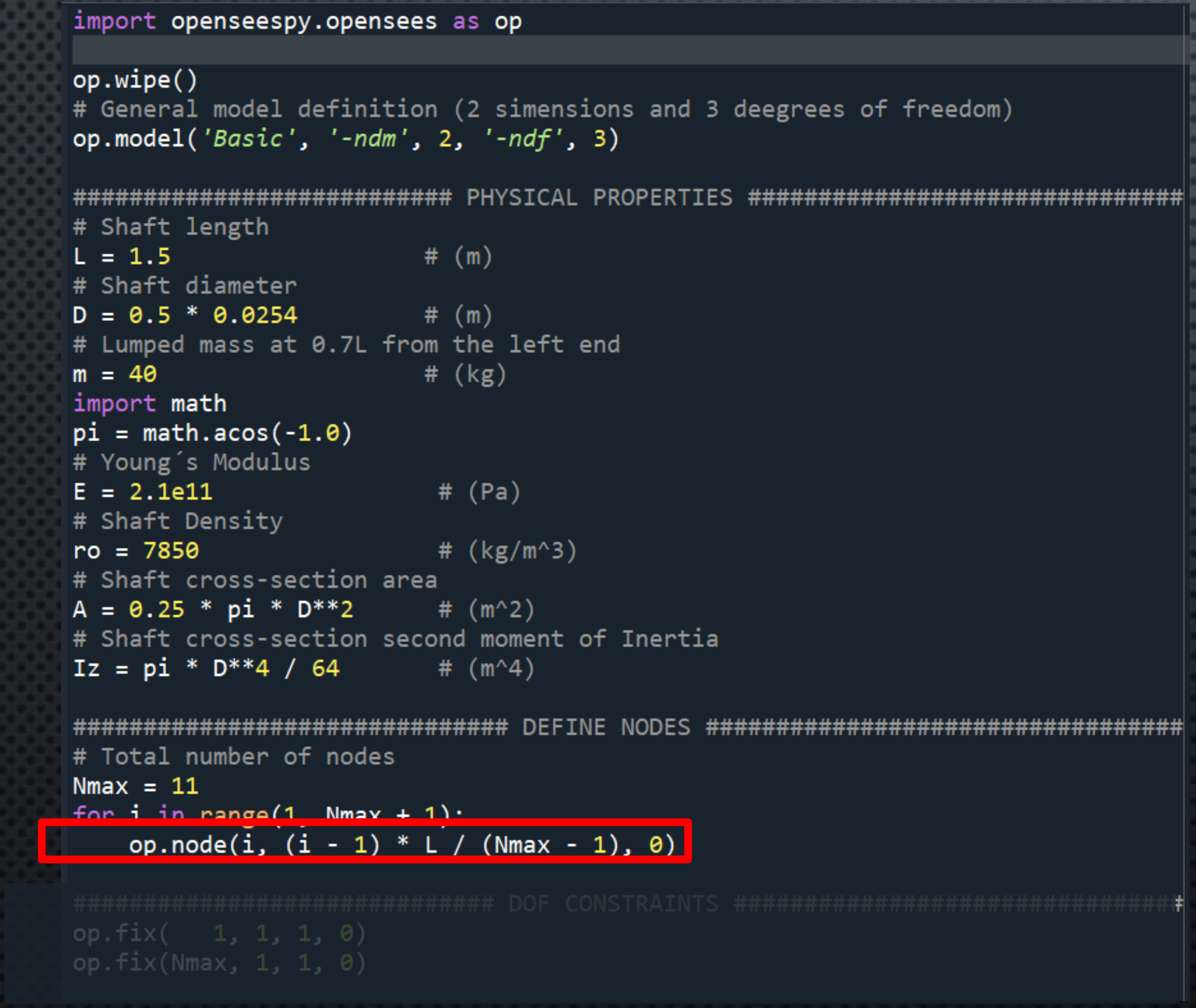


```python
import openseespy.opensees as op

op.wipe()
# General model definition (2 simensions and 3 deegrees of freedom)
op.model('Basic', '-ndm', 2, '-ndf', 3)

########################### PHYSICAL PROPERTIES ################################
# Shaft length
L = 1.5                   # (m)
# Shaft diameter
D = 0.5 * 0.0254          # (m)
# Lumped mass at 0.7L from the left end
m = 40                    # (kg)
import math
pi = math.acos(-1.0)
# Young´s Modulus
E = 2.1e11                 # (Pa)
# Shaft Density
ro = 7850                  # (kg/m^3)
# Shaft cross-section area
A = 0.25 * pi * D**2       # (m^2)
# Shaft cross-section second moment of Inertia
Iz = pi * D**4 / 64        # (m^4)

############################### DEFINE NODES ###################################
# Total number of nodes
Nmax = 11
for i in range(1, Nmax + 1):
    op.node(i, (i - 1) * L / (Nmax - 1), 0)

############################## DOF CONSTRAINTS #################################
op.fix(   1, 1, 1, 0)
op.fix(Nmax, 1, 1, 0)
```

For complex models or those with a significant number of nodes and elements, it is good practice to define variables containing geometry and material attributes. This helps both, you and others understand the analysis more effectively.

Even though this analysis is relatively simple, we have defined the physical properties of the model as provided in the problem description.

NODE DEFINITION



```
import openseespy.opensees as op

op.wipe()
# General model definition (2 simensions and 3 deegrees of freedom)
op.model('Basic', '-ndm', 2, '-ndf', 3)

########################### PHYSICAL PROPERTIES ###############################
# Shaft length
L = 1.5                  # (m)
# Shaft diameter
D = 0.5 * 0.0254         # (m)
# Lumped mass at 0.7L from the left end
m = 40                   # (kg)
import math
pi = math.acos(-1.0)
# Young´s Modulus
E = 2.1e11                # (Pa)
# Shaft Density
ro = 7850                 # (kg/m^3)
# Shaft cross-section area
A = 0.25 * pi * D**2      # (m^2)
# Shaft cross-section second moment of Inertia
Iz = pi * D**4 / 64       # (m^4)

############################### DEFINE NODES ##################################
# Total number of nodes
Nmax = 11
for i in range(1, Nmax + 1):
    op.node(i, (i - 1) * L / (Nmax - 1), 0)

############################## DOF CONSTRAINTS ###############################
op.fix(   1, 1, 1, 0)
op.fix(Nmax, 1, 1, 0)
```

After defining the model, the next critical step is to define the nodes. This is accomplished using the node command, which follows this syntax:

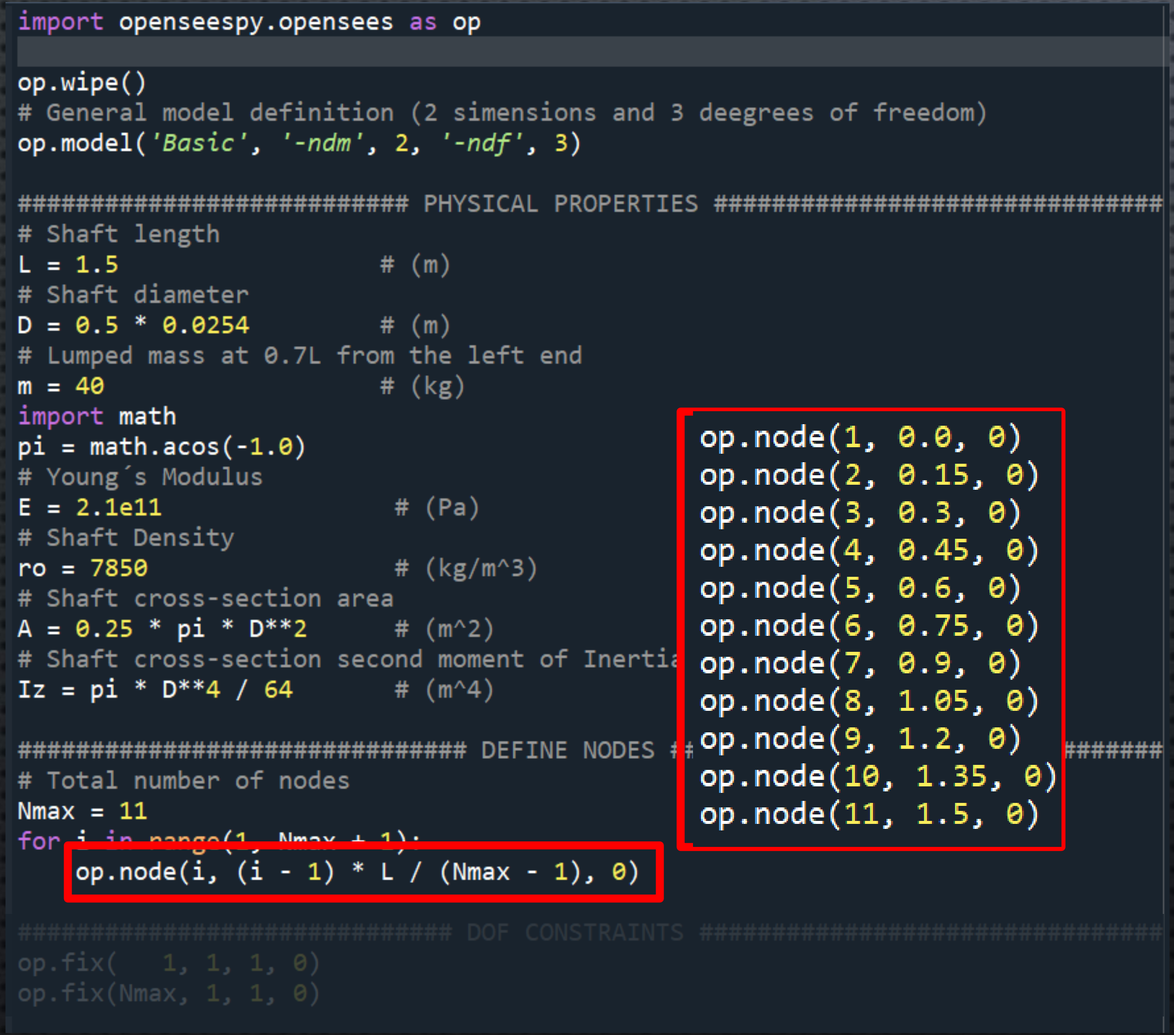
node(nodeTag, *crds, '-ndf', ndf, '-mass', mass, '-disp', *disp, 'vel', *vel, '-accel', *accel)

Where:

- nodeTag is a **unique** identifier for each node.
- *crds represents the spatial coordinates of the nodes (consistent with the dimension defined in the model command).

The remaining command options are optional but refer to nodal properties, including nodal degree of freedom ('-ndf', ndf), nodal mass ('-mass', mass´), nodal displacement ('-disp', *disp), nodal velocity ('vel', *vel) and nodal acceleration ('-accel', *accel)

NODE DEFINITION (CONTINUED)



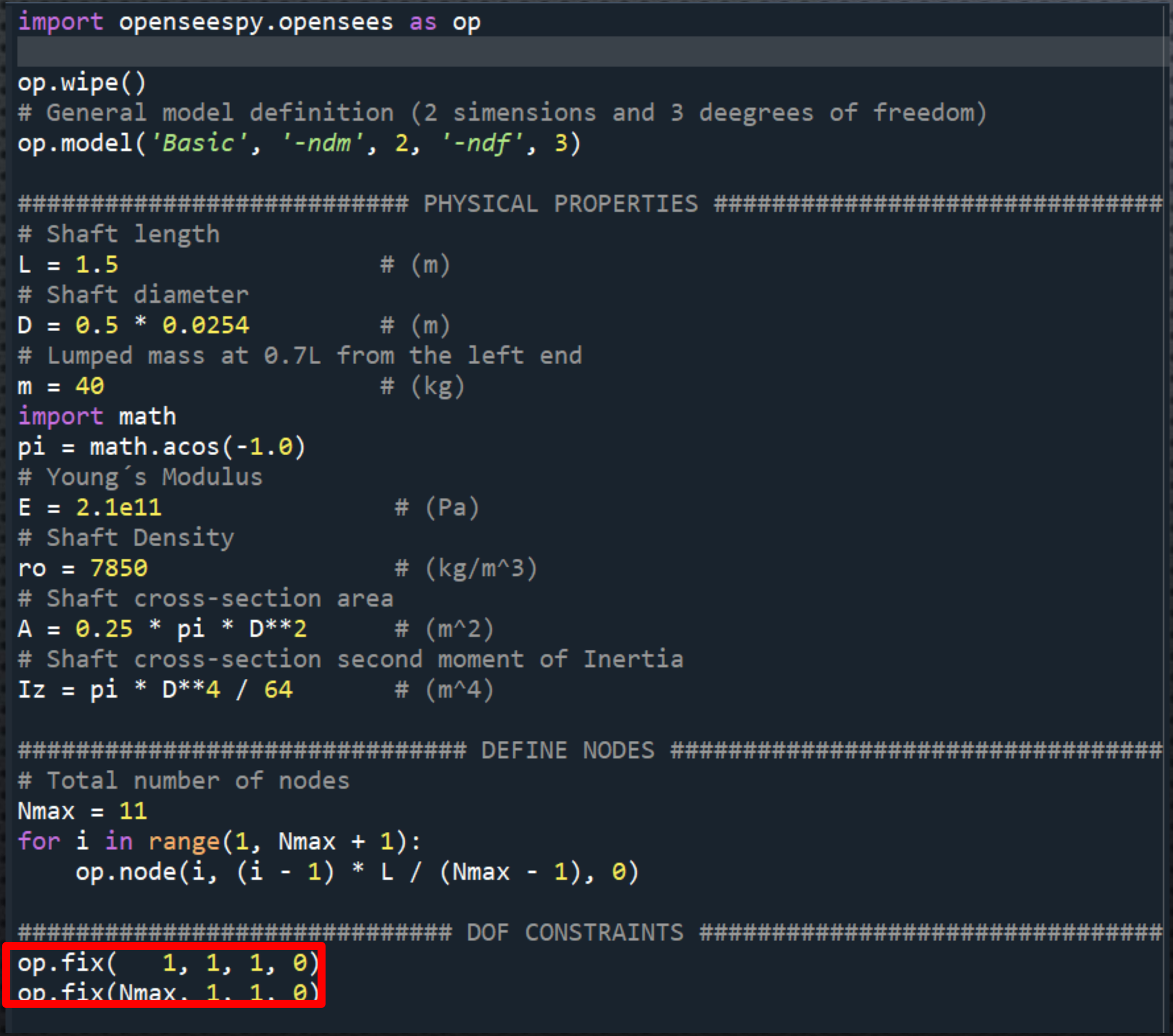
```
import openseespy.opensees as op

op.wipe()
# General model definition (2 simensions and 3 deegrees of freedom)
op.model('Basic', '-ndm', 2, '-ndf', 3)

########################### PHYSICAL PROPERTIES ################################
# Shaft length
L = 1.5                    # (m)
# Shaft diameter
D = 0.5 * 0.0254           # (m)
# Lumped mass at 0.7L from the left end
m = 40                     # (kg)
import math
pi = math.acos(-1.0)
# Young´s Modulus
E = 2.1e11                  # (Pa)
# Shaft Density
ro = 7850                   # (kg/m^3)
# Shaft cross-section area
A = 0.25 * pi * D**2        # (m^2)
# Shaft cross-section second moment of Inertia
Iz = pi * D**4 / 64         # (m^4)

############################### DEFINE NODES ##                         #######
# Total number of nodes
Nmax = 11
for i in range(1, Nmax + 1):
    op.node(i, (i - 1) * L / (Nmax - 1), 0)

############################## DOF CONSTRAINTS ##############################
op.fix(   1, 1, 1, 0)
op.fix(Nmax, 1, 1, 0)
```

```
op.node(1, 0.0, 0)
op.node(2, 0.15, 0)
op.node(3, 0.3, 0)
op.node(4, 0.45, 0)
op.node(5, 0.6, 0)
op.node(6, 0.75, 0)
op.node(7, 0.9, 0)
op.node(8, 1.05, 0)
op.node(9, 1.2, 0)
op.node(10, 1.35, 0)
op.node(11, 1.5, 0)
```

The node definitions were originally accomplished iteratively using a for command. However, It´s important to note that this approach is equivalent to defining each node one by one, as illustrated in the figure.

Furthermore, it is worth mentioning that, in this 2D model (as specified in the model command), only two coordinates are required for each node.

RESTRAINTS



```
import openseespy.opensees as op

op.wipe()
# General model definition (2 simensions and 3 deegrees of freedom)
op.model('Basic', '-ndm', 2, '-ndf', 3)

########################### PHYSICAL PROPERTIES ################################
# Shaft length
L = 1.5                  # (m)
# Shaft diameter
D = 0.5 * 0.0254         # (m)
# Lumped mass at 0.7L from the left end
m = 40                   # (kg)
import math
pi = math.acos(-1.0)
# Young´s Modulus
E = 2.1e11                # (Pa)
# Shaft Density
ro = 7850                 # (kg/m^3)
# Shaft cross-section area
A = 0.25 * pi * D**2      # (m^2)
# Shaft cross-section second moment of Inertia
Iz = pi * D**4 / 64       # (m^4)

############################### DEFINE NODES #################################
# Total number of nodes
Nmax = 11
for i in range(1, Nmax + 1):
    op.node(i, (i - 1) * L / (Nmax - 1), 0)

############################## DOF CONSTRAINTS ###############################
op.fix(   1, 1, 1, 0)
op.fix(Nmax, 1, 1, 0)
```

The fix command is used to impose constraints on a specific node in the model. The syntax for this command is aas follows:

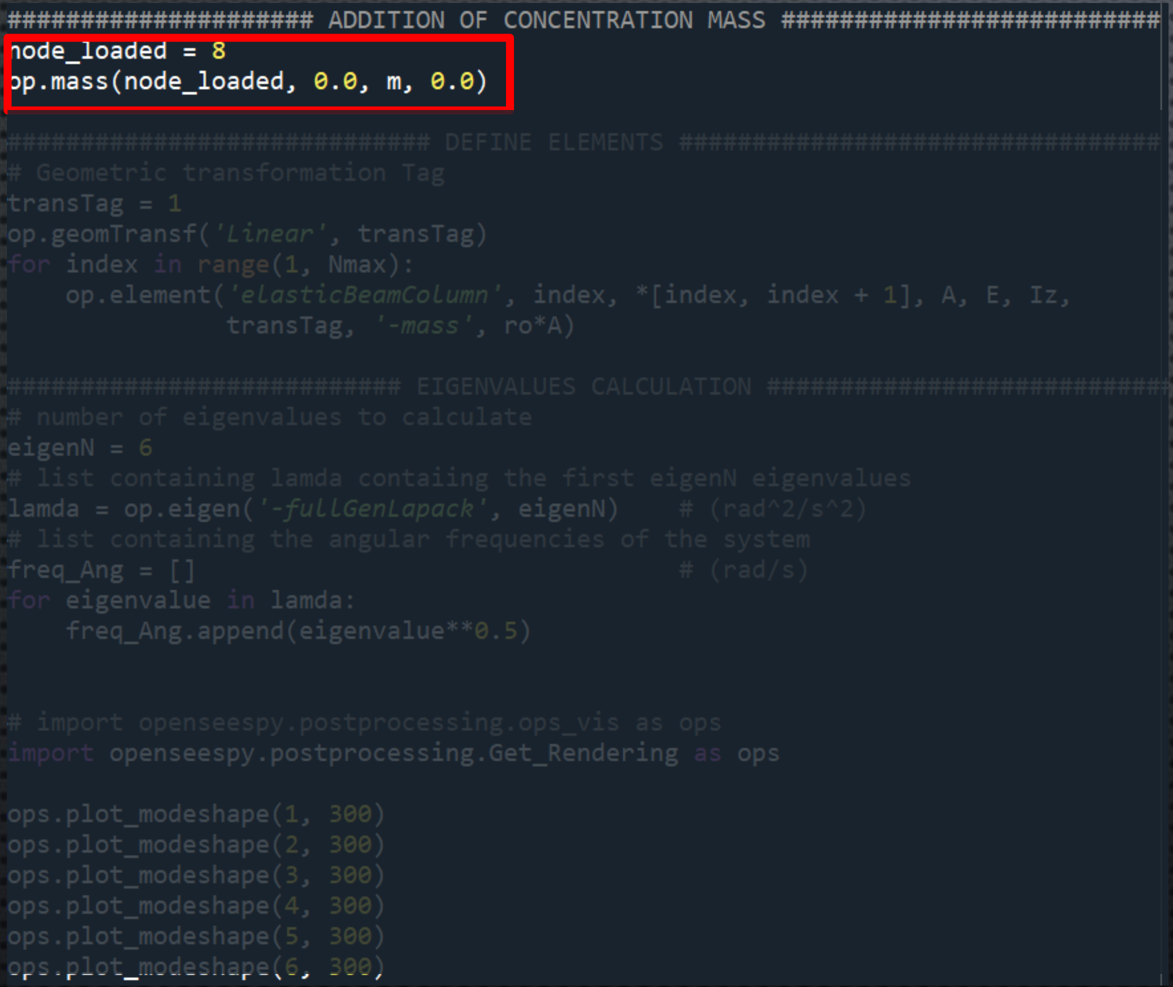
fix(nodeTag, *constrValues)

Where:

- Node tag represents the node's unique identifier, which was defined during node creation.
- *constrValues are Boolean constraint values (0 for free degrees of freedom (dof) and 1 for fixed dof).

In this example, only the initial node (i=1) and the last node (i = Nmax) are restrained. This means that both nodes have fixed displacements in the x and y directions, while allowing free rotation about the z-axis.

LUMPED MASS



```
##################### ADDITION OF CONCENTRATION MASS ############################
node_loaded = 8
op.mass(node_loaded, 0.0, m, 0.0)

############################# DEFINE ELEMENTS #################################
# Geometric transformation Tag
transTag = 1
op.geomTransf('Linear', transTag)
for index in range(1, Nmax):
    op.element('elasticBeamColumn', index, *[index, index + 1], A, E, Iz,
               transTag, '-mass', ro*A)

########################### EIGENVALUES CALCULATION ############################
# number of eigenvalues to calculate
eigenN = 6
# list containing lamda containg the first eigenN eigenvalues
lamda = op.eigen('-fullGenLapack', eigenN)    # (rad^2/s^2)
# list containing the angular frequencies of the system
freq_Ang = []                                 # (rad/s)
for eigenvalue in lamda:
    freq_Ang.append(eigenvalue**0.5)


# import openseespy.postprocessing.ops_vis as ops
import openseespy.postprocessing.Get_Rendering as ops

ops.plot_modeshape(1, 300)
ops.plot_modeshape(2, 300)
ops.plot_modeshape(3, 300)
ops.plot_modeshape(4, 300)
ops.plot_modeshape(5, 300)
ops.plot_modeshape(6, 300)
```

The mass command is necessary in this example due to the presence of a concentrated mass at one of the nodes. However, distributed mass related to the density of the shaft, will be defined along with the elements, as we will demonstrate later.

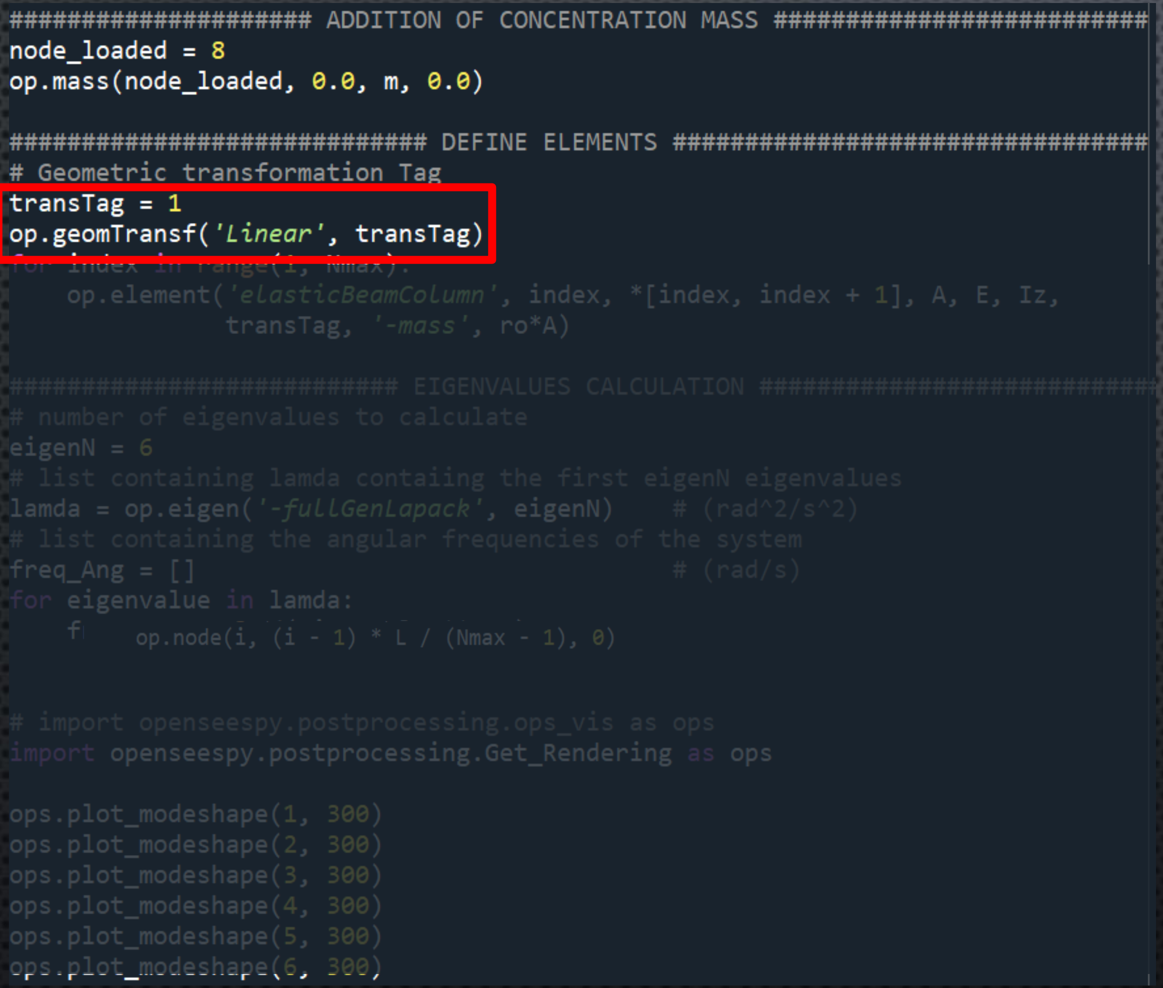
The syntax for the mass command is as follow:

mass(nodeTag, *massValues)

Where:

- node Tag refers to the corresponding node ID.
- *massValues are the values of the mass in the respective degree of freedom.

GEOMETRIC TRANSFORMATION



```
###################### ADDITION OF CONCENTRATION MASS ###########################
node_loaded = 8
op.mass(node_loaded, 0.0, m, 0.0)

############################## DEFINE ELEMENTS ###################################
# Geometric transformation Tag
transTag = 1
op.geomTransf('Linear', transTag)
for index in range(1, Nmax):
    op.element('elasticBeamColumn', index, *[index, index + 1], A, E, Iz,
               transTag, '-mass', ro*A)

############################ EIGENVALUES CALCULATION ############################
# number of eigenvalues to calculate
eigenN = 6
# list containing lamda contaiing the first eigenN eigenvalues
lamda = op.eigen('-fullGenLapack', eigenN)    # (rad^2/s^2)
# list containing the angular frequencies of the system
freq_Ang = []                                 # (rad/s)
for eigenvalue in lamda:
    f    op.node(i, (i - 1) * L / (Nmax - 1), 0)


# import openseespy.postprocessing.ops_vis as ops
import openseespy.postprocessing.Get_Rendering as ops

ops.plot_modeshape(1, 300)
ops.plot_modeshape(2, 300)
ops.plot_modeshape(3, 300)
ops.plot_modeshape(4, 300)
ops.plot_modeshape(5, 300)
ops.plot_modeshape(6, 300)
```

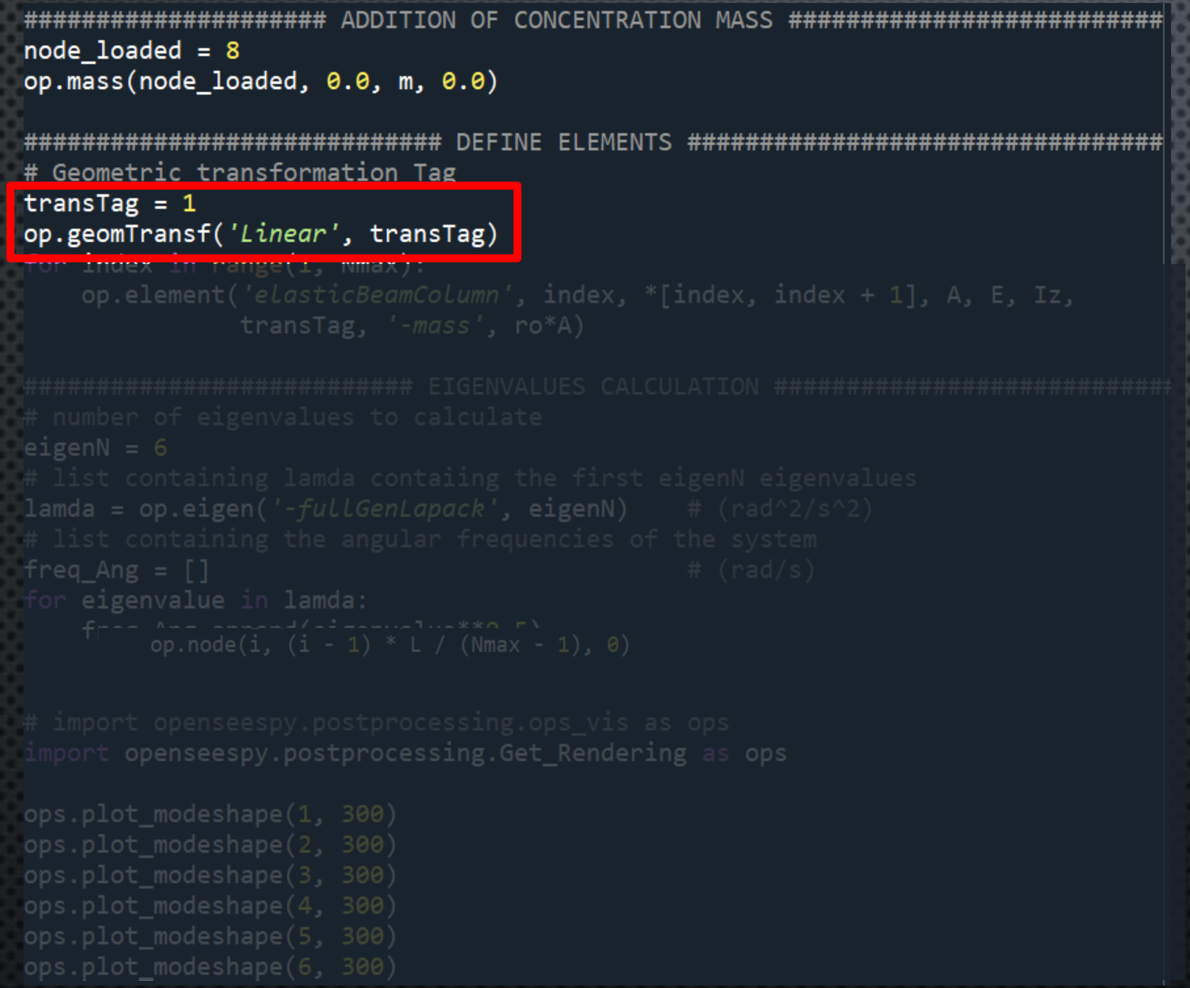
Since we plan to use an elastic beam element to model the shaft, a geometric transformation is necessary to convert the local coordinate system to the global coordinate system. The syntax for this command is as follows:

geoTrasnf(transfType, transTag, *trasnfArgs)

Where:

- transfType represents the type of trasnfoamrtion (Linear, Pdelta or corotational).
- trasnfTag is a unique transformation ID.
- transfArgs is a list of arguments for the geometric trasnfromation.

GEOMETRIC TRANSFORMATION (CONTINUED)



```
##################### ADDITION OF CONCENTRATION MASS ###########################
node_loaded = 8
op.mass(node_loaded, 0.0, m, 0.0)

############################## DEFINE ELEMENTS ##################################
# Geometric transformation Tag
transTag = 1
op.geomTransf('Linear', transTag)
for index in range(1, Nmax):
    op.element('elasticBeamColumn', index, *[index, index + 1], A, E, Iz,
               transTag, '-mass', ro*A)

########################### EIGENVALUES CALCULATION ############################
# number of eigenvalues to calculate
eigenN = 6
# list containing lamda contaiing the first eigenN eigenvalues
lamda = op.eigen('-fullGenLapack', eigenN)    # (rad^2/s^2)
# list containing the angular frequencies of the system
freq_Ang = []                                  # (rad/s)
for eigenvalue in lamda:
    freq_Ang.append(eigenvalue**0.5)
        op.node(i, (i - 1) * L / (Nmax - 1), 0)


# import openseespy.postprocessing.ops_vis as ops
import openseespy.postprocessing.Get_Rendering as ops

ops.plot_modeshape(1, 300)
ops.plot_modeshape(2, 300)
ops.plot_modeshape(3, 300)
ops.plot_modeshape(4, 300)
ops.plot_modeshape(5, 300)
ops.plot_modeshape(6, 300)
```

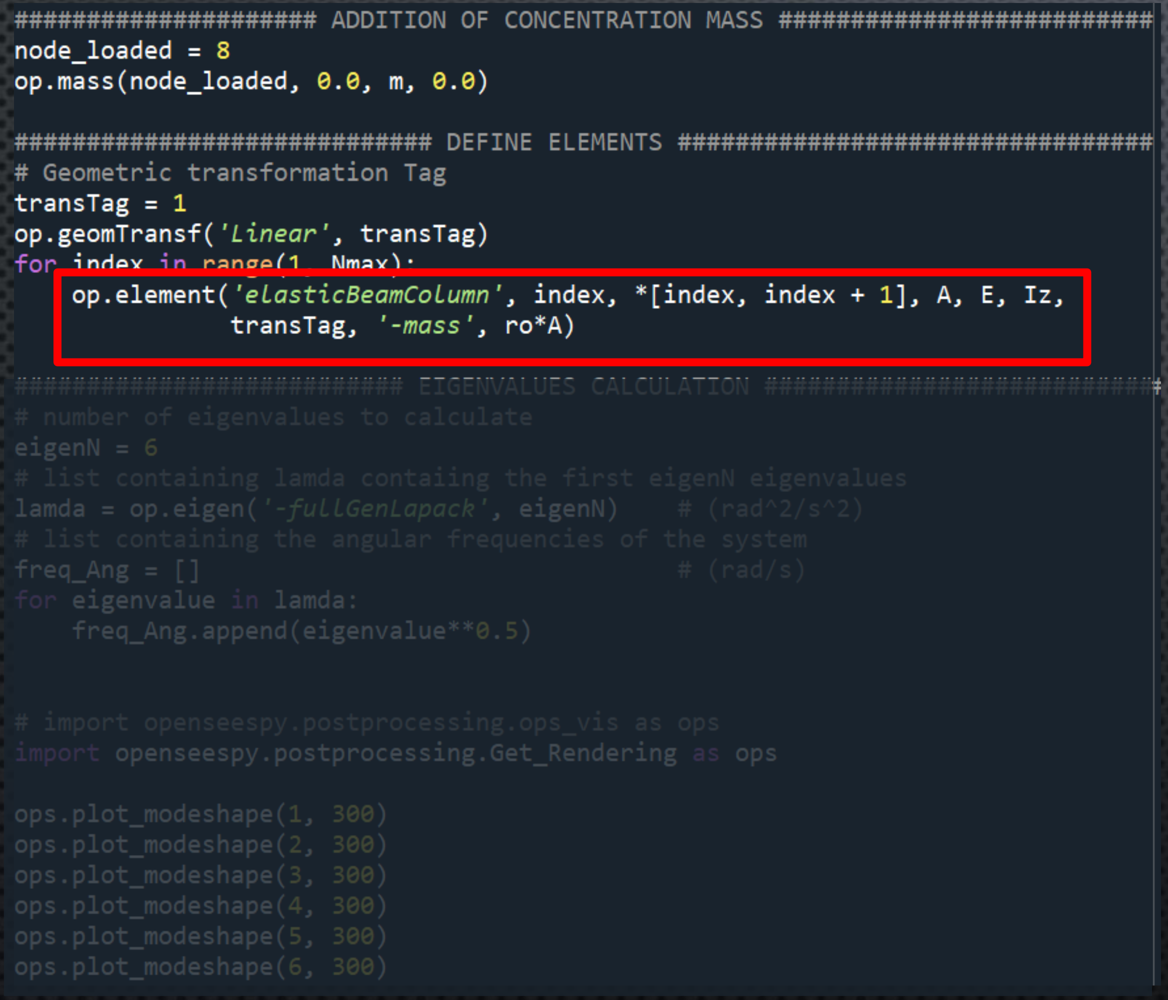
In this example, a linear transformation was used. When using the geoTransf command, the general format resembles the following:

geoTransf('Linear', tranfTag, *vecxz, '-jntOffset', *dI, *dJ)

Where:

- Vecxz represents the x,y and z components of the vector used to define the local x-z plane of the local coordinate system (applicable to 3D beam element)
- dI and dJ are joint offset values (optional in this context but required for 3D models).

ELEMENT DEFINITION



```
##################### ADDITION OF CONCENTRATION MASS ###########################
node_loaded = 8
op.mass(node_loaded, 0.0, m, 0.0)

############################# DEFINE ELEMENTS ##################################
# Geometric transformation Tag
transTag = 1
op.geomTransf('Linear', transTag)
for index in range(1, Nmax):
    op.element('elasticBeamColumn', index, *[index, index + 1], A, E, Iz,
               transTag, '-mass', ro*A)
########################## EIGENVALUES CALCULATION ############################
# number of eigenvalues to calculate
eigenN = 6
# list containing lamda contaiing the first eigenN eigenvalues
lamda = op.eigen('-fullGenLapack', eigenN)    # (rad^2/s^2)
# list containing the angular frequencies of the system
freq_Ang = []                                 # (rad/s)
for eigenvalue in lamda:
    freq_Ang.append(eigenvalue**0.5)


# import openseespy.postprocessing.ops_vis as ops
import openseespy.postprocessing.Get_Rendering as ops

ops.plot_modeshape(1, 300)
ops.plot_modeshape(2, 300)
ops.plot_modeshape(3, 300)
ops.plot_modeshape(4, 300)
ops.plot_modeshape(5, 300)
ops.plot_modeshape(6, 300)
```

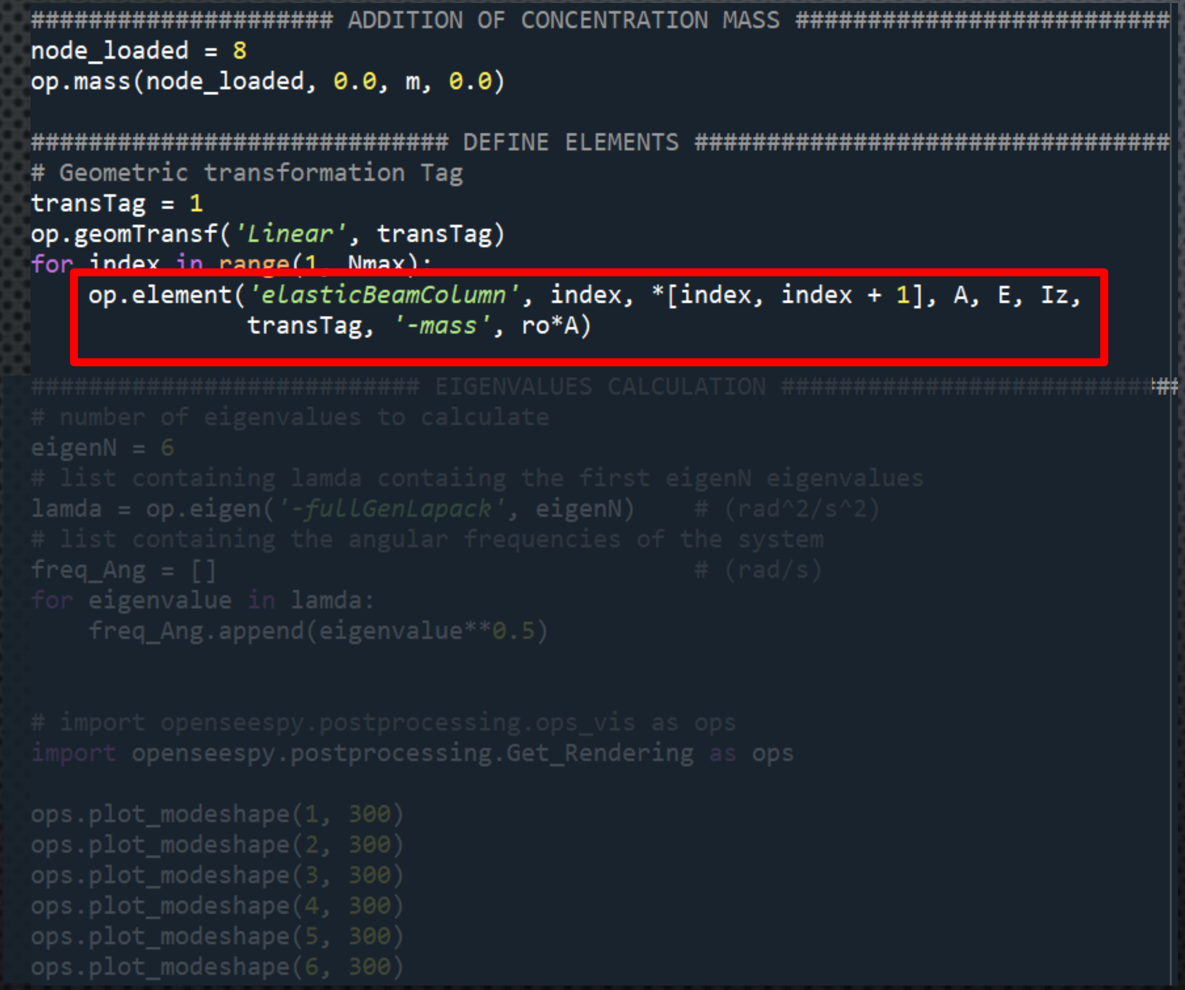
The preceding steps were essential to define the element. In this example, we have defined elastic BeamColumn elements. However, it is worth noticing that the list of elements can vary including zero-length, truss, joint, link elements, and more.

The syntax used for the elasticBeamColumn is described as follows:

Element(‘elasticBeamColumn’, eleTag, *eleNodes, Area, E_mod, Iz, transfTag, <’-mass’, mass>, <’-cMass’>, <‘-release’, releaseCode>)

Where:
- eleTag refers to the element ID.
- *eleNodes are the two nodes that define the element.

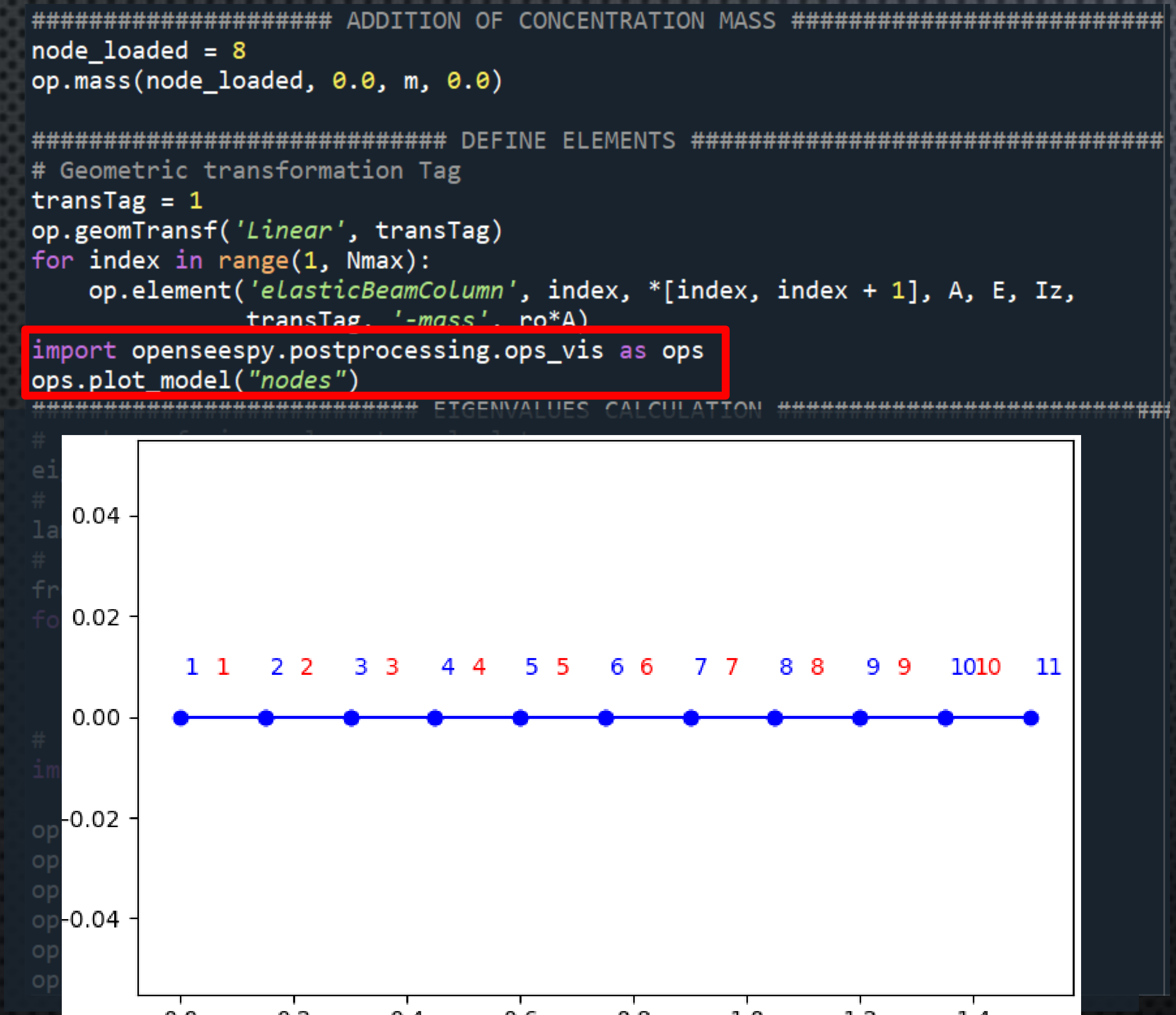
ELEMENT DEFINITION (CONTINUED)



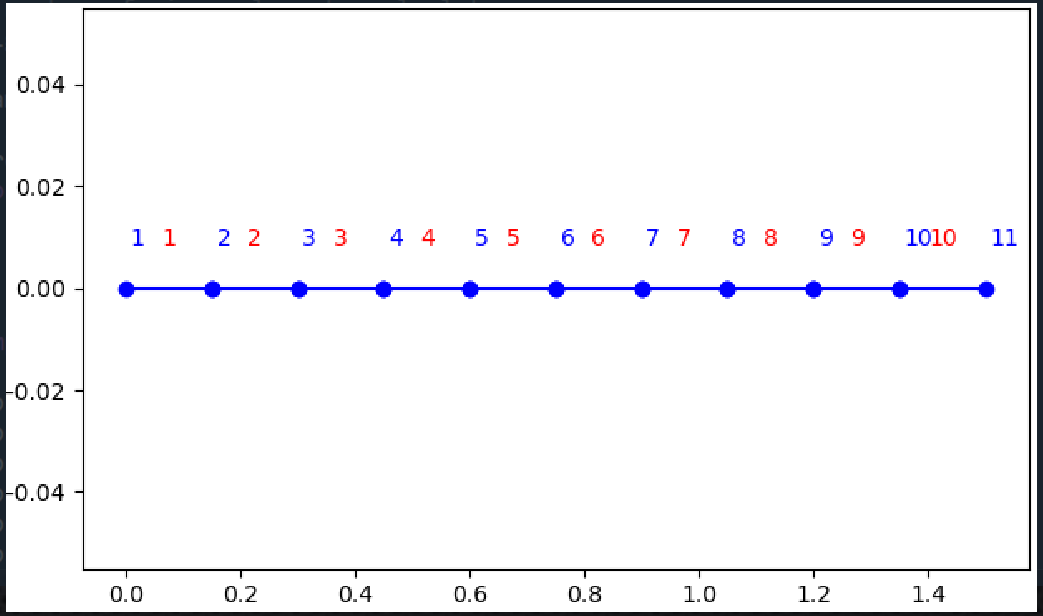
```
##################### ADDITION OF CONCENTRATION MASS ###########################
node_loaded = 8
op.mass(node_loaded, 0.0, m, 0.0)

############################# DEFINE ELEMENTS ##################################
# Geometric transformation Tag
transTag = 1
op.geomTransf('Linear', transTag)
for index in range(1, Nmax):
    op.element('elasticBeamColumn', index, *[index, index + 1], A, E, Iz,
               transTag, '-mass', ro*A)

########################## EIGENVALUES CALCULATION ##########################
# number of eigenvalues to calculate
eigenN = 6
# list containing lamda contaiing the first eigenN eigenvalues
lamda = op.eigen('-fullGenLapack', eigenN)    # (rad^2/s^2)
# list containing the angular frequencies of the system
freq_Ang = []                                 # (rad/s)
for eigenvalue in lamda:
    freq_Ang.append(eigenvalue**0.5)


# import openseespy.postprocessing.ops_vis as ops
import openseespy.postprocessing.Get_Rendering as ops

ops.plot_modeshape(1, 300)
ops.plot_modeshape(2, 300)
ops.plot_modeshape(3, 300)
ops.plot_modeshape(4, 300)
ops.plot_modeshape(5, 300)
ops.plot_modeshape(6, 300)
```

- Area, E_mod and Iz are the cross sectional area, Young’s modulus and second moment of Inertia, respectively.
- transfTag is the transformation tag.
- ’-mass’ is the identifier for the mass per unit length.
- mass is the value of the mass per unit length
- Release is an optional value that represents the release conditions.

NODES & ELEMENTS VISUALIZATION



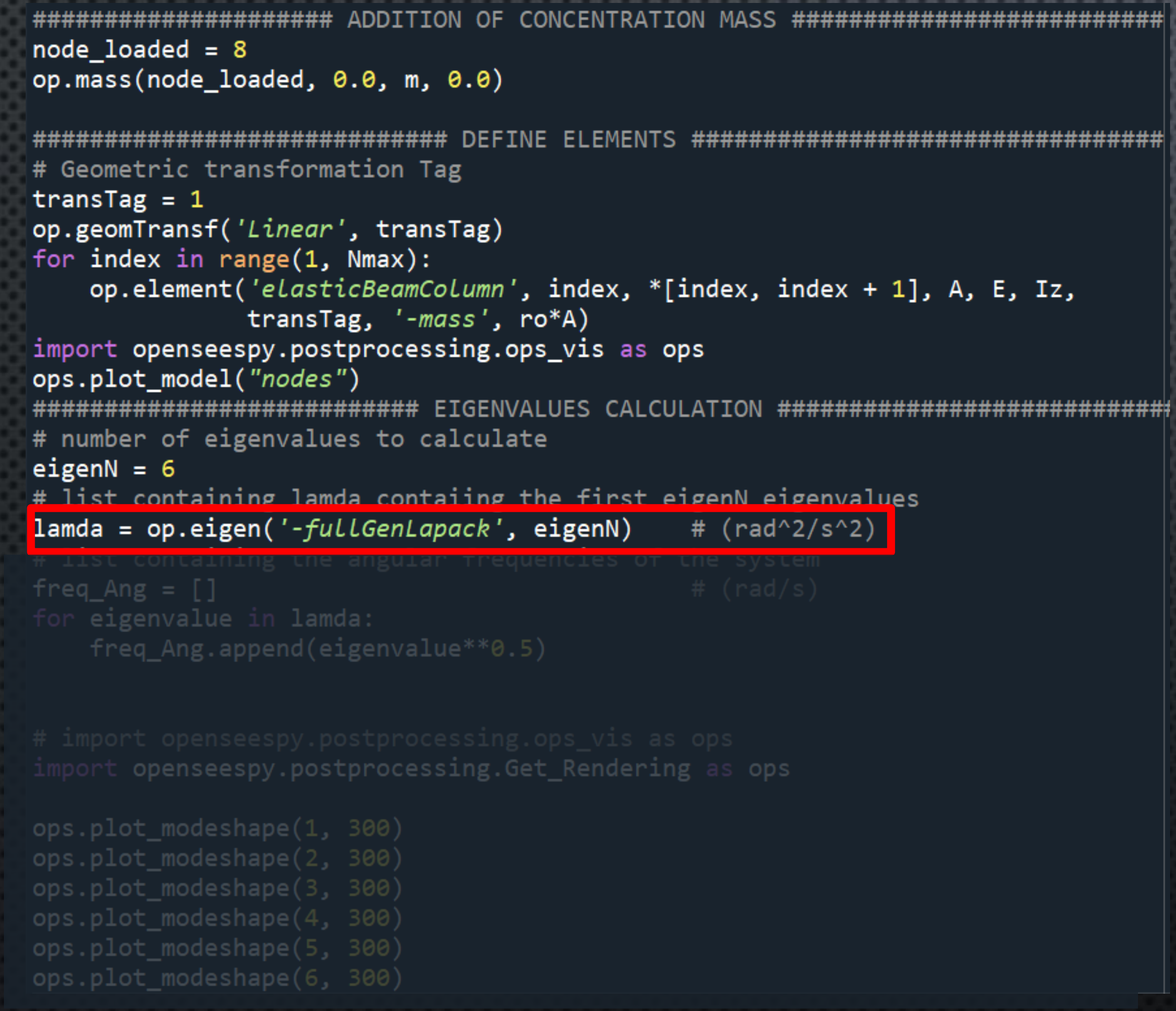
```
#################### ADDITION OF CONCENTRATION MASS ##########################
node_loaded = 8
op.mass(node_loaded, 0.0, m, 0.0)

############################## DEFINE ELEMENTS ##################################
# Geometric transformation Tag
transTag = 1
op.geomTransf('Linear', transTag)
for index in range(1, Nmax):
    op.element('elasticBeamColumn', index, *[index, index + 1], A, E, Iz,
               transTag, '-mass', ro*A)
import openseespy.postprocessing.ops_vis as ops
ops.plot_model("nodes")
########################### EIGENVALUES CALCULATION ###########################
#
ei
#
la
#
fr
fo


#
im

op
op
op
op
op
op
```



Now that we have already defined the elements and nodes in the model, it might be valuable to obtain a graphical representation to confirm whether the intended model definition was achieved as intended.

To proceed with it, we can import the **openseespostprocessing** library. Then, we can use the plot_model command to generate a graphical representation of the model.

EIGENVALUES CALCULATION



```
##################### ADDITION OF CONCENTRATION MASS ##########################
node_loaded = 8
op.mass(node_loaded, 0.0, m, 0.0)

############################# DEFINE ELEMENTS #################################
# Geometric transformation Tag
transTag = 1
op.geomTransf('Linear', transTag)
for index in range(1, Nmax):
    op.element('elasticBeamColumn', index, *[index, index + 1], A, E, Iz,
               transTag, '-mass', ro*A)
import openseespy.postprocessing.ops_vis as ops
ops.plot_model("nodes")
########################### EIGENVALUES CALCULATION ############################
# number of eigenvalues to calculate
eigenN = 6
# list containing lamda containing the first eigenN eigenvalues
lamda = op.eigen('-fullGenLapack', eigenN)    # (rad^2/s^2)
# list containing the angular frequencies of the system
freq_Ang = []                                 # (rad/s)
for eigenvalue in lamda:
    freq_Ang.append(eigenvalue**0.5)


# import openseespy.postprocessing.ops_vis as ops
import openseespy.postprocessing.Get_Rendering as ops

ops.plot_modeshape(1, 300)
ops.plot_modeshape(2, 300)
ops.plot_modeshape(3, 300)
ops.plot_modeshape(4, 300)
ops.plot_modeshape(5, 300)
ops.plot_modeshape(6, 300)
```

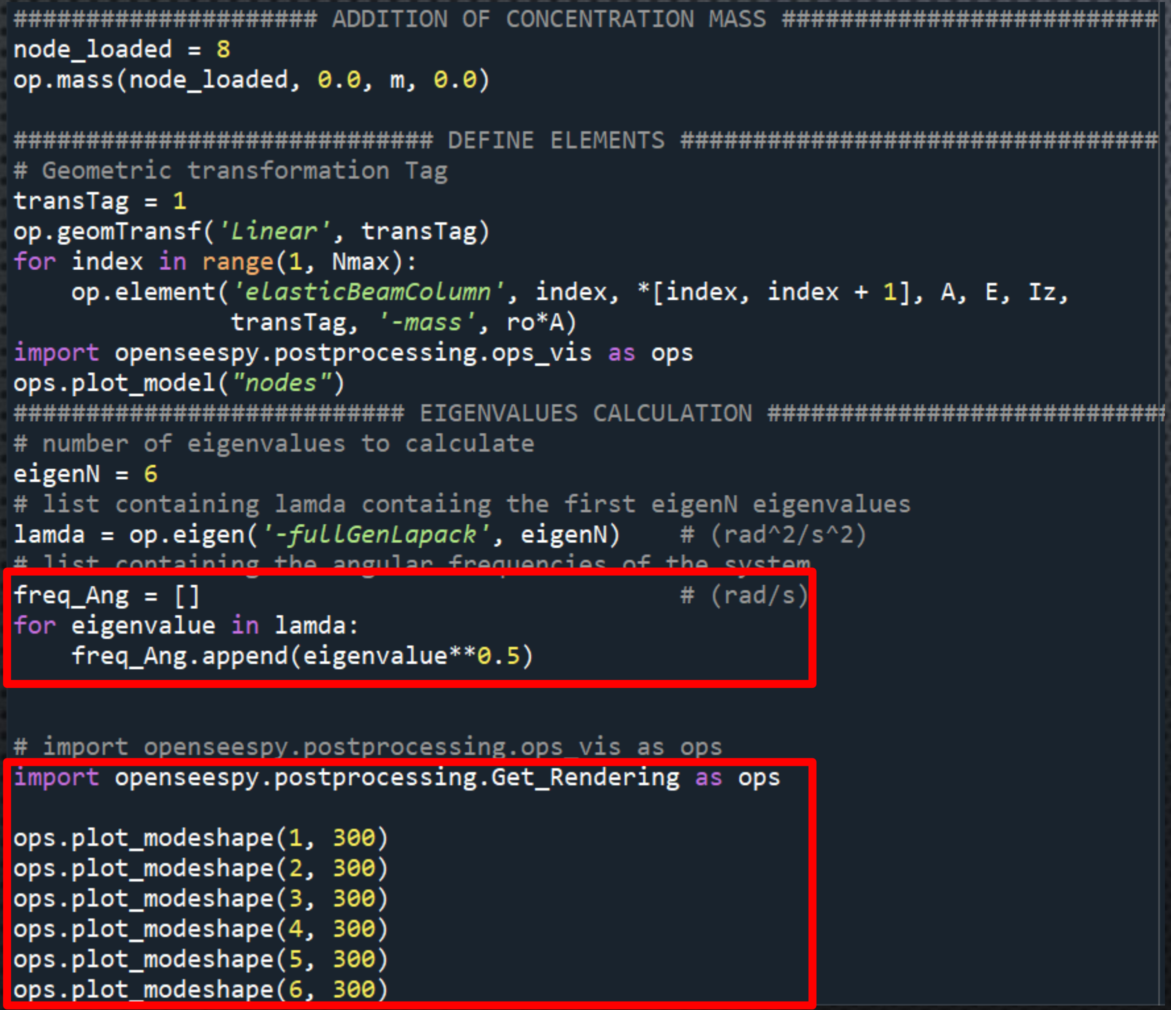
After the elements are properly defined, we proceed to calculate the eigenvalues using the eigen command. This command, in a general form, is represented as follows:

eigen(solver, numEigenValues)

Where:

- solver is the type of solver to use (optional parameter), with two available types ('-genBandArpack' and '-fullGenLapack').
- numEigenValues is the number of eigenvalues to calculate.

EIGENVALUES CALCULATION AND MODES REPRESENTATION



```
##################### ADDITION OF CONCENTRATION MASS ##########################
node_loaded = 8
op.mass(node_loaded, 0.0, m, 0.0)

############################# DEFINE ELEMENTS #################################
# Geometric transformation Tag
transTag = 1
op.geomTransf('Linear', transTag)
for index in range(1, Nmax):
    op.element('elasticBeamColumn', index, *[index, index + 1], A, E, Iz,
               transTag, '-mass', ro*A)
import openseespy.postprocessing.ops_vis as ops
ops.plot_model("nodes")
########################### EIGENVALUES CALCULATION ###########################
# number of eigenvalues to calculate
eigenN = 6
# list containing lamda containg the first eigenN eigenvalues
lamda = op.eigen('-fullGenLapack', eigenN)     # (rad^2/s^2)
# list containing the angular frequencies of the system
freq_Ang = []                                  # (rad/s)
for eigenvalue in lamda:
    freq_Ang.append(eigenvalue**0.5)


# import openseespy.postprocessing.ops_vis as ops
import openseespy.postprocessing.Get_Rendering as ops

ops.plot_modeshape(1, 300)
ops.plot_modeshape(2, 300)
ops.plot_modeshape(3, 300)
ops.plot_modeshape(4, 300)
ops.plot_modeshape(5, 300)
ops.plot_modeshape(6, 300)
```

In the final step, once we have obtained the eigenvalues, we proceed to calculate the angular frequencies.

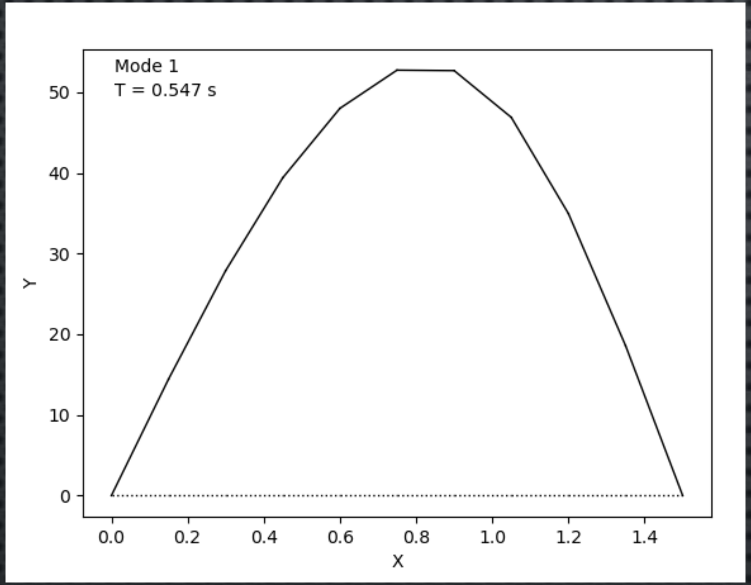
Moreover, to visualize the eigenmodes, we import once again **openseespy.postprocessing**. In this case, we utilize the plot_modeshape command with the following syntax:

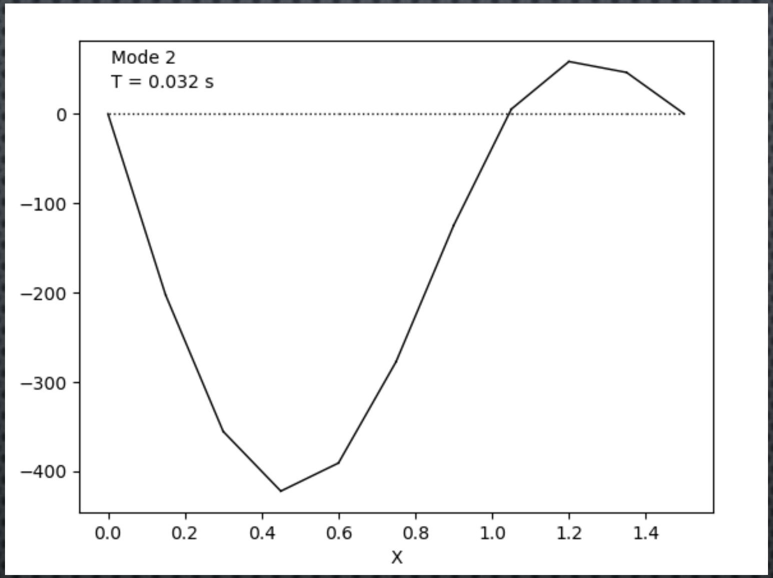
plot_modeshape(modeNumber, <scale>, <Model="modeName">)

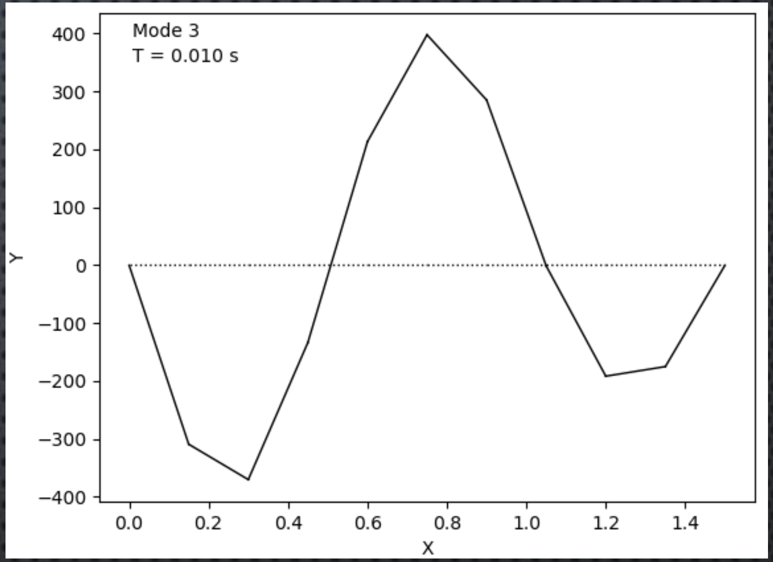
Where:

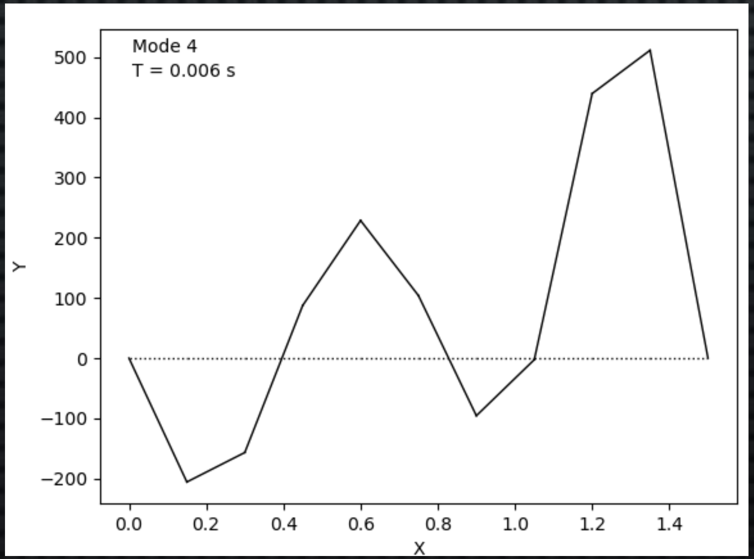
- modeNumber corresponds to the mode number you want to visualize.
- <scale> (optional) can be used to adjust the scale of the visualization.
- <Model="modeName">' (optional) specifies the mode name for the model."

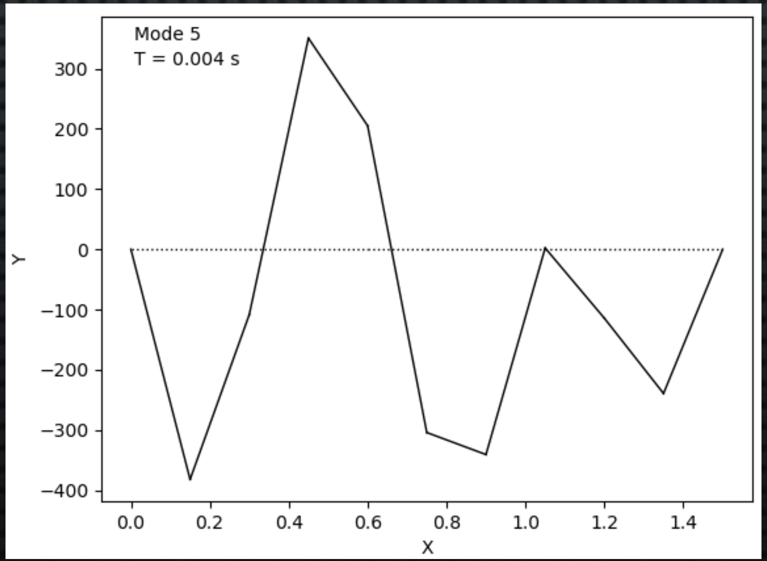
EIGENMODES VISUALIZATION

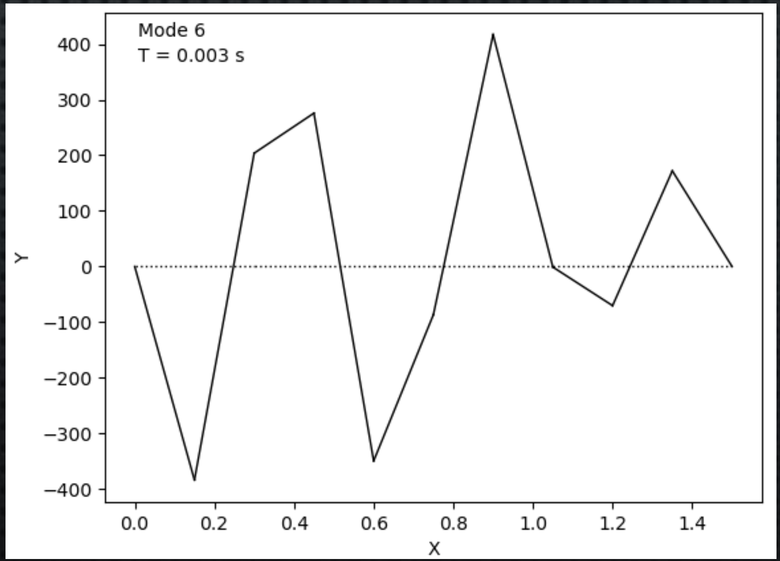












REFERENCES

For more detailed information about these commands, you can refer to the official documentation at:

https://openseespydoc.readthedocs.io/en/latest/index.html

http://opensees.berkeley.edu/wiki/index.php