planesections

Release 1.2.2

CSlotboom

May 03, 2024

CONTENTS

PlaneSections is a lightweight finite element beam bending library built on libraries like PyNite or OpenSeesPy. The goal of PlaneSections is to quickly run beam anayses, and create outputs to document structural calculations. **Note, results are only stored at nodes specified by the user - all intermediate values in plots are linearly interpolated**

The following website documents all classes and functions the user can access in PlaneSections. The core of the program is the beam class in the builder module. This classes will encapuslate all of the contains all information for loading serve as interfaces for class is the beam, and the user interacts with this class to run their analysis.

The Builder module is used to create the beam and objects that interact with the beam (nodes, loads, etc.). The Analysis module is used to analyze the beam using PyNite or OpenSeesPy and document the results. The Diagram module is used to plot representations of the beam. The Postprocess module is used to plot outputs of the analysis, including force diagrams and deflections

Note that the core classes and API are complete, but development is still in progress. Expect some sytax changes before final release, however deprication warnings will be given for breaking changes.

Install using:

```
pip -m install planesections
```

Install with optional dependancies for opensees solver using:

```
pip -m install planesections[opensees]
```

A simple script is shown below:

```
import planesections as ps
import numpy as np

# Define node locations, and support conditions
L = 5
beam = ps.newEulerBeam2D(L)

# Define beam and fixities
pinned = [1,1,0]
beam.setFixity(L*0.1, pinned, label = '1')
beam.setFixity(L*0.9, pinned)

# Define loads
Pz = -1
beam.addVerticalLoad(0, Pz, label = 'A')
beam.addVerticalLoad(L*0.5, 2*Pz, label = 'B')
beam.addVerticalLoad(L, Pz, label = 'C')
beam.addDistLoadVertical(0, L, Pz)
beam.addDistLoadVertical(1, L*0.3, 5*Pz)

# Plot the beam diagram
ps.plotBeamDiagram(beam)

# Run the analysis
analysis = ps.PyNiteAnalyzer2D(beam)
analysis.runAnalysis()

# Plot the SFD and BMD
ps.plotShear(beam)
ps.plotMoment(beam)
```

CHAPTER

ONE

BUILDER

These classes and functions are used by the user to build the representation of the beam. The beam will store information using seperate section, node, and load classes. While the user can make these classes independently, most of these classes will be accessed throught methods in the beam class.

1. *Beam Sections*
2. *Beam Nodes*
3. *Beam*
4. *Beam Loads*

1.1 Beam Sections

These classes represent the possible beam sections that can be used. Sections are used to determine propreties about the beam such as it's elastic modulus, area, or moment of Inertia.

1. *SectionBasic*
2. *SectionRectangle*

1.1.1 SectionBasic

class planesections.section.**SectionBasic**(*E: float = 1*, *G: float = 1*, *A: float = 1*, *Iz: float = 1*, *Iy: float = 1*, *J: float = 1*)

Bases: `Section`

A basic section that contains the global propreties of the beam section, without any geometry. It's assume the section is elastic.

A: float = 1

E: float = 1

G: float = 1

Iy: float = 1

Iz: float = 1

J: float = 1

1.1.2 SectionRectangle

class planesections.section.**SectionRectangle**(*E: float = 200000000000, d: float = 1, w: float = 1, G: float | None = None, units: str = 'm'*)

Bases: Section

Represents a elastic Rectangular section. Iz and A are calcualted using the beam width and height.

https://www.structx.com/Shape_Formulas_024.html

E: float = 200000000000

G: float = None

d: float = 1

units: str = 'm'

w: float = 1

1.2 Beam Nodes

These are the classes used to represent nodes on the beam. All loads, fixities, and labels are linked to the node they act on.

1. *2D Node*
2. *3D Node*

1.2.1 2D Node

A 2D node with three degrees of freedom, ux, uy, and rotation.

class planesections.builder.**Node2D**(*x: float, fixity: list | str | Fixity, label: str = ''*)

Bases: Node

Represents a node 2D. Nodes have labels and IDs and fixities. The Label is a name the user assigns to the node and will be displayed in plots.

The ID is a unique name that OpenSees will read. ID - 1 will be the position in the beam node array. As new nodes are added, the IDs will be sorted and updated so that they are always increasing from left to right.

Parameters

- **x** (*float*) – The postion of the node.
- **fixity** (*[list, str, Fixity]*) – In 2D, the fixity can be input as either a fixtiy object, a string from the variable NAMED_RELEASES_2D, or A list of the input fixities for each possible degree of freedom.

  Each node will have three degree of freedoms; [x, y, θ] 1 represents a fixed condition, 0 represents a free conditon.

  The passed object can be a fixity object, a string e.x. 'pinned', or a list of integers, e.x. [1,1,0] gives a pin conneciton that's fixed in x/y but free in rotation.
- **label** (*str, optional*) – A name for the node. This can be displayed in the plots. The default is ''.

getFixityType()

Returns the type of beam fixity for supported 2D fixities. Currently only free, roller, pinned, and fixed are supported.

1.2.2 3D Node

A 3D node with six degrees of freedom, ux, uy, uz, rx, ry, rz.

class planesections.builder.**Node3D**(*x: float*, *fixity: list | str | Fixity*, *label: str = ''*)

Bases: Node

Represents a node 3D. Nodes have labels and IDs and fixities. The Label is a name the user assigns to the node and will be displayed in plots.

The ID is a unique name that OpenSees will read. ID - 1 will be the position in the beam node array. As new nodes are added, the IDs will be sorted and updated so that they are always increasing from left to right.

Parameters

- **x** (*float*) – The postion of the node.
- **fixity** (*fixity, list*) – In 3D, the fixity can be input as either a fixtiy object, or a list of the input fixities for each possible degree of freedom. Each node will have six degree of freedoms; [x, y, :math:`theta`x] 1 represents a fixed condition, 0 represents a free conditon. e.x. [1,1,0,1,1,1] A pin conneciton that's fixed in x/y and fixed all in rotation DOF.
- **label** (*str, optional*) – A name for the node. This can be displayed in the plots. The default is ''.

getFixityType()

Unsupported for 3D beams.

1.3 Beam

These classes and functions are used to make and interact with the beam.

1. *Beam*
2. *Fixities*
3. *Eulerbeam*
4. *New Euler Beam*
5. *New Simply Supported Beam*

1.3.1 Beam

The main class that all 2D beam classes inherit from. Contains many methods that allow the user to create nodes, forces, and other points of interest.

class planesections.builder.**Beam**

Bases: object

A representation of a beam object, that can be used to define information about basic beams. Units must form a consist unit basis for FEM analysis.

The base Beam class isn't used by the user, the inherited classes Node2D and Node3D are used instead.

Fmax(*index*)

get the maximum and minimum internal force for teh beam along the appropriate axis. 0:x, 1:y, 2:z (In 3D, 3:rx, 4:ry, 5:rz)

addDistLoad(*x1: float*, *x2: float*, *distLoad: float*, *label: str = ''*)

Adds a distributed load to the model. The load is defined between two locations, x1 and x2, in the model. If nodes exist at these locations, then the load is definied between those existing nodes. If there are no nodes at these locations, then nodes are added to the model. Old loads at this point are deleted.

Parameters

- **x1** (*float*) – Start point of distributed load.
- **x2** (*float*) – Start point of distributed load.
- **distLoad** (*array*) – The distributed load.

  In 2D has the form [Fx (axial force), Fy (shear force)]

  In 3D has the form [Fx (axial force), Fy (shear force), Fz (shear force)]
- **label** (*str*) – A optional label for the force.

addDistLoadHorizontal(*x1: float*, *x2: float*, *qx: float*, *label: str = ''*)

Adds a distributed load to the model. The load is defined between two locations, x1 and x2, in the model. If nodes exist at these locations, then the load is definied between those existing nodes. If there are no nodes at these locations, then nodes are added to the model. Old loads at this point are deleted.

Parameters

- **x1** (*float*) – Start point of distributed load.
- **x2** (*float*) – Start point of distributed load.
- **qx** (*float*) – A constantly distributed axial force.
- **label** (*str*) – A optional label for the force.

addDistLoadVertical(*x1: float*, *x2: float*, *qy: float*, *label: str = ''*)

Adds a distributed load to the model. The load is defined between two locations, x1 and x2, in the model. If nodes exist at these locations, then the load is definied between those existing nodes. If there are no nodes at these locations, then nodes are added to the model. Old loads at this point are deleted.

Parameters

- **x1** (*float*) – Start point of distributed load.
- **x2** (*float*) – Start point of distributed load.
- **qy** (*float*) – A constantly distributed vertical force.
- **label** (*str*) – A optional label for the force.

addHorizontalLoad(*x: float*, *Px: float*, *label: str = ''*, *labelNode=False*)

Adds a horizontal point load at the model at location x. If no node exists at position x, a new node is added. Old loads are deleted. x : float

The x location to add force at.

Px

[float] The magnitude of the vertical load to be added at x.

label

[str, optional] The label of the input node. labels are displayed in the plots. The default is ‘’.

addLabel(*x: float*, *label: str*, *sort: bool = True*)

Adds a label to the beam at the coordinate in question. If a node exists at this location the label is added to it. If no node exists at location x, a new node is added. The new node will have default fixity.

Parameters

- **x** (*float*) – The x coordinate of the node.
- **label** (*str, optional*) – A name for the node. This can be displayed in the plots. The default is ‘’.
- **sort** (*bool, optional*) – A switch which turns on or off sorting of the nodes after a lable is added. The default value is True, which sorts the nodes.

Returns

flag – returns 0 if a existing node has been updated, 1 if a new node is added, and -1 if the process failed.

Return type

int

addLinLoad(*x1: float*, *x2: float*, *linLoad: list[list]*, *label: str = ''*)

Adds a load that linearly varies between two input values. The load is defined between two locations, x1 and x2. If nodes exist at these locations, then the load is definied between those existing nodes. If there are no nodes at these locations, then nodes are added to the model. Old loads at this point are deleted.

Parameters

- **x1** (*float*) – Start point of distributed load.
- **x2** (*float*) – Start point of distributed load.
- **linLoad** (*array*) – The distributed load. The loads given are the maximum of the distributed load

  In 2D has the form [[qx_start, qx_end], [qy_start, qy_end]], where x is an axial force and y is a shear force.

  In 3D has the form [[qx_start, qx_end],
  [qy_start, qy_end], [qz_start, qz_end],]

  Where x is an axial force and y is shear force, and z is out of plane shear force..

label

[str] A optional label for the force.

addLinLoadHorizontal(*x1: float*, *x2: float*, *qx: list[float]*, *label: str = ''*)

Adds a linear load to the model is defined between two locations, x1 and x2, in the model. If nodes exist at these locations, then the load is definied between those existing nodes. If there are no nodes at these locations, then nodes are added to the model. Old loads at this point are deleted.

Parameters

- **x1** (*float*) – Start point of distributed load.
- **x2** (*float*) – Start point of distributed load.
- **qx** (*float*) – A list of y values to linearly distribute between.
- **label** (*str*) – A optional label for the force.

addLinLoadVertical(*x1: float*, *x2: float*, *qy: list[float]*, *label: str = ''*, ***kwargs*)

Adds a linear load to the model. The load is defined between two locations in the model, x1 and x2. If nodes exist at these x1 or x2, then the load is definied between those existing nodes. If there are no nodes at these locations, then nodes are added to the model. Old loads at this point are deleted.

Parameters

- **x1** (*float*) – Start point of distributed load.
- **x2** (*float*) – Start point of distributed load.
- **qy** (*float*) – The peak load for a linearly disributed vertical load.
- **label** (*str*) – A optional label for the force.

addMoment(*x: float*, *M: float*, *label: str = ''*, *labelNode=False*)

Adds a moment ot the model at location x. If no node exists at position x, a new node is added. Old loads at this point are deleted. .. todo:: State which direction positive is.

Parameters

- **x** (*float*) – The x location to add a moment at.
- **M** (*float*) – The magnitude of the moment to be added at x.
- **label** (*str, optional*) – The label of the input node. labels are displayed in the plots. The default is ‘’.

addNode(*x: float*, *fixity: list | str | Fixity | None = None*, *label: str = ''*, *sort: bool = True*)

Adds a new node to the beam. Keyword arguments are passed to the node. See *Node2D* for more details

Parameters

- **x** (*float*) – The x coordinate of the node.
- **fixity** (*Fixity, list*) – A fixity object, or a list of the input fixities for each possible degree of freedom. 2D nodes have three degree of freedoms; [x, y, θ] 3D nodes have six degree of freedoms; [x, y, z, θ_x, θ_y, θ_z] For each degree of freedom, 1 represents a fixed condition, 0 represents a free conditon. e.x.

  [1,1,0] - A 2D connection that’s fixed in x/y but free in rotation.

  [1,1,0,0,0,1] - A 3D connection that’s fixed in x/y and θ_z .
- **label** (*str, optional*) – A name for the node. This can be displayed in the plots. The default is ‘’.
- **sort** (*bool, optional*) – A toggle that turns on or off node sorting as new nodes are added. Nodes are sorted after each new node as added, this can be toggled off to inprove performance. However, nodes must be sorted before the analysis is run.

Returns

flag – returns 0 if a existing node has been updated, 1 if a new node is added, and -1 if the process failed.

Return type

int

addNodes(*xCoords: list[float]*, *fixities: list[list | str | Fixity] | None = None*, *labels: list[str] | None = None*)

Adds several new nodes to the beam at the same time. The nodes in question are added at the x coordinates in the model. Nodes are sorted at the end of the process

Parameters

- **xCoords** (*list of float*) – A list of the x coordinates to be added to the model.
- **fixities** (*list of fixity or booliean, optional*) – A fixity object, or a list of the input fixities for each possible degree of freedom. 2D nodes have three degree of freedoms; [x, y, θ] 3D nodes have six degree of freedoms; [x, y, z, θ_x, θ_y, θ_z] For each degree of freedom, 1 represents a fixed condition, 0 represents a free conditon. e.x.

  [1,1,0] - A 2D connection that's fixed in x/y but free in rotation.

  [1,1,0,0,0,1] - A 3D connection that's fixed in x/y and θ_z .
- **label** (*list[str], optional*) – A list of the labels for each node. labels are displayed in the plots. The default is ''.

addPointLoad(*x: float*, *pointLoad: list*, *label: str = ''*, *labelNode=False*)

Adds a load ot the model at location x. If a node exists at the current location, the old load value is overwritten. Old loads are deleted, and the node is relabled. Can represent objects in 2D or 3D.

Parameters

- **x** (*float*) – The location of the load.
- **pointLoad** (*list*) – A list of the forces. For a 2D beam has form [Fx, Fy, M]. For a 3D beam has form [Fx, Fy, Fz, Mx, My, Mz].

  New Load 1:
  [0., 10., 0.] A vertical load of 10 is applied in beam units.

  New Load 2:
  [0., 0., 13] A moment of 13 is applied in beam units.
- **label** (*str, optional*) – The label of the input node. labels are displayed in the plots. The default is ''.

addVerticalLoad(*x: float*, *Py: float*, *label: str = ''*, *labelNode=False*)

Adds a vertical load to the model at location x. If no node exists at position x, a new node is added. Old loads at this point are deleted.

Parameters

- **x** (*float*) – The x location to add force at.
- **Py** (*float*) – The magnitude of the vertical load to be added at x.
- **label** (*str, optional*) – The label of the input node. labels are displayed in the plots. The default is ''.

getDOF()

Returns the number of degrees of Freedom at each point in the beam.

getLength()

Returns the length of the beam.

Returns
The beam length.

Return type
float

getNodeIDs()

Gets all of the node IDs.

Returns
IDs – a list of all of the node IDs in the model.

Return type
list[int]

getNodes()

getxLims()

Returns the of the beam.

Returns
A list with the left most and right most point.

Return type
list[float]

setFixity(*x: float*, *fixity: list[list | Fixity]*, *label=None*)

Sets the the model fixity at locaiton x. If the node exists, update it. If the node doesn't exist, then a new node will be added

Parameters

- **x** (*float*) – The x coordinant of the noded to be modified/added.
- **fixity** (*list, Fixity*) – A fixity object, or a list of the input fixities for each possible degree of freedom. 2D nodes have three degree of freedoms; [x, y, θ] 3D nodes have six degree of freedoms; [x, y, z, θ_x, θ_y, θ_z] For each degree of freedom, 1 represents a fixed condition, 0 represents a free conditon. e.x.

  [1,1,0] - A 2D connection that's fixed in x/y but free in rotation.

  [1,1,0,0,0,1] - A 3D connection that's fixed in x/y and θ_z .
- **label** (*str, optional*) – The label of the input node. labels are displayed in the plots. The default is ''.

1.3.2 Fixities

A custom fixity class. Note all fixities can be replaced by a list of appropriate size with a 1 if the DOF is fixed, and 0 if it is free.

class planesections.builder.**Fixity**(*name: str*, *fixityValues: list[int]*)

Bases: object

class planesections.builder.**FixityTypes2D**

Bases: object

Used to generate possible fixity types. Currently the supported types for 2D fixities are free, roller, pinned, and fixed. The Fixity class can be used to dispatch each of these objects with it's relevant "get" methods.

Note each fixity types will all be set equal to the same object as opposed to new objects being created.

Takes input of:

- 'free'
- 'roller'
- 'pinned'
- 'fixed'

fixed = <Fixity type fixed with [1, 1, 1].>

free = <Fixity type free with [0, 0, 0].>

classmethod getFixed()

Returns a fixed support.

classmethod getFree()

Returns a free support.

classmethod getPinned()

Returns a pinned support.

classmethod getRoller()

Returns a roller support.

pinned = <Fixity type pinned with [1, 1, 0].>

releaseNames = ['free', 'roller', 'pinned', 'fixed']

releaseTypes = [[0, 0, 0], [0, 1, 0], [1, 1, 0], [1, 1, 1]]

roller = <Fixity type roller with [0, 1, 0].>

types2D = {'fixed': <Fixity type fixed with [1, 1, 1].>, 'free': <Fixity type free with [0, 0, 0].>, 'pinned': <Fixity type pinned with [1, 1, 0].>, 'roller': <Fixity type roller with [0, 1, 0].>}

1.3.3 Eulerbeam

class planesections.builder.**EulerBeam**(*xcoords: list | None = None*, *fixities: list | None = None*, *labels: list | None = None*, *section=None*, *dimension='2D'*)

Bases: *Beam*

A creates a 2D/3D Euler beam. Information about the beam is stored in a mesh of nodes across the beam that are added by the user. Note that only output information at the nodes will be contained in the analysis.

The units of the beam must form a consistent unit base for FEM

Inherits from the base *Beam* class.

Parameters

- **xcoords** (*list, optional*) – The x coodinates of nodes along the beam the beam. The default is [], which starts with no nodes.
- **fixity**(*list of Fixity, or list of lists*) – A list of fixity objects, or A list of the input fixities for each possible degree of freedom. 2D nodes have three degree of freedoms; [x, y, θ] 3D nodes have six degree of freedoms; [x, y, z, θ_x, θ_y, θ_z] For each degree of freedom, 1 represents a fixed condition, 0 represents a free conditon. e.x.

  [1,1,0] - A 2D connection that's fixed in x/y but free in rotation.

  [1,1,0,0,0,1] - A 3D connection that's fixed in x/y and θ_z .
- **labels** (*list, optional*) – A list of labels for each node. The default is [], which gives no label to each node.
- **section** (*Section2D, optional*) – The section to use in the anaysis. The default uses SectionBasic2D().

property Mmax

property Vmax

getBMD()

Returns the left and right bending moment at each node in the model. Because the diagram is discrete, left and right forces must be used to capture discontinuties.

Returns

- **xcoords** (*array*) – the x coordinants, has vale for x and y.
- **Moment** (*array*) – the output left and right moment at each node

getInternalForce(*index*)

Returns the left and right internal forces each node in the model for the input force type. Because the diagram is discrete, left and right forces must be used to capture discontinuties.

Parameters

- **index** (*int*)
- **use** (*The index of the force type to*) – 0: axial force 1: shear force 2: bending

Returns

- **xcoords** (*array*) – the x coordinants, has vale for x and y.
- **force** (*array*) – the output force at each node

getMaterialPropreties()

Returns the material properties of a section.

In 2D returns E, G, A, Iz

In 3D returns E, G, A, Iy, Iz, J

Returns
DESCRIPTION.

Return type
list

getMoment()

Depricated. See getBMD.

Returns

- **xcoords** (*array*) – the x coordinants, has vale for x and y.
- **Moment** (*array*) – the output left and right moment at each node

getSFD()

Returns the left and right shear force at each node in the model. Because the diagram is discrete, left and right forces must be used to capture discontinuties.

Returns

- **xcoords** (*array*) – the x coordinants, has vale for x and y.
- **Moment** (*array*) – the output left and right moment at each node

property reactionDict

property reactions

1.3.4 New Euler Beam

planesections.builder.**newEulerBeam**(*x2*, *x1=0*, *meshSize=101*, *section=None*, *dimension='2D'*)

Initializes a new 2D Euler beam. The beam will have no fixities or labels.

Parameters

- **x2** (*float*) – The end position of the beam. If no x1 is provided, this is also the length of the beam
- **x1** (*float, optional*) – The start position of the beam. The default is 0.
- **meshSize** (*int, optional*) – The mesh size for the beam. This many nodes will be added between the points x1 and x2. The default is 101, which divides the beam into 100 even sections..
- **section** (*Section2D, optional*) – The section to use in the anaysis. The default uses SectionBasic2D().

Returns

EulerBeam2D – The the beam intialized with the mesh of points between x1 and x2.

Return type

EulerBeam

1.3.5 New Simply Supported Beam

planesections.builder.**newSimpleEulerBeam**(*x2*, *x1=0*, *meshSize=101*, *q=0*, *section=None*, *dimension='2D'*)

Initializes a new simply supported Euler beam with a distributed load. The beam will have no fixities or labels.

Parameters

- **x2** (*float*) – The end position of the beam. If no x1 is provided, this is also the length of the beam
- **x1** (*float, optional*) – The start position of the beam. The default is 0.
- **meshSize** (*int, optional*) – The mesh size for the beam. This many nodes will be added between the points x1 and x2. The default is 101, which divides the beam into 100 even sections..
- **q** (*float, optional*) – The distributed load on the simply supported beam.
- **section** (*Section2D, optional*) – The section to use in the anaysis. The default uses SectionBasic2D().

Returns

EulerBeam2D – The the beam intialized with the mesh of points between x1 and x2.

Return type

EulerBeam

1.4 Beam Loads

These classes store information about beam loads. Only point loads, line loads, and linearly varying loads are supported.

1. *Point Load*
2. *Constant Element Load*
3. *Linear Element Load*

1.4.1 Point Load

class planesections.builder.**PointLoad**(*P*, *x*, *nodeID=None*, *label=''*)

Bases: `object`

Representes a point load at locaiton x

Parameters

- **P** (*list[float]*) – List of forces. In 2D, as form [Px, Py, M].
- **x** (*float*) – The location of the point load.
- **nodeID** (*int*) – List of forces in [Px, Py].
- **label** (*str, optional*) – A label for the elment load. The default is ‘’.

getPosition()

nodeID = None

1.4.2 Constant Element Load

class planesections.builder.**EleLoadDist**(*x1: float*, *x2: float*, *distLoad: list*, *label: str = ''*)

Bases: `object`

Representes a constantly distrubted element load between two points x1 x2. For 2D elements, distributed loads can either px or py.

Parameters

- **x1** (*float*) – The start position.
- **x2** (*float*) – The end position.
- **distLoad** (*list*) – For 2D, a List of forces in [Px, Py].
- **label** (*str, optional*) – A label for the elment load. The default is ‘’.

1.4.3 Linear Element Load

class planesections.builder.**EleLoadLinear**(*x1: float*, *x2: float*, *linLoad: list*, *label: str = ''*)

Bases: object

Represents a linearly distrubted element load between two points x1 x2, where the load increase from x1 to x2. The direction of the load can be toggled so the high point is at x1 instead of x2.

Parameters

- **x1** (*float*) – The start position.
- **x2** (*float*) – The end position.
- **distLoad** (*list*) – For 2D, a List of forces in [Px, Py].
- **label** (*str, optional*) – A label for the elment load. The default is ‘’.

checkInRange(*s*)

Checks if a x value is in the range x1/x2 of the force.

getLoadComponents(*s1*, *s2*, *q*)

Gets the load at two intermedia points, s1/s2.

CHAPTER

TWO

ANALYSIS

These classes are used to analyze the beam. OpenSeesPy is used for all analysis.

1. *outputRecorder*
2. *OpenSeesAnalyzer2D*
3. *OpenSeesAnalyzer3D*
4. *PyNiteAnalyzer2D*

2.1 outputRecorder

class planesections.analysis.openSees.**OutputRecorder**

Bases: ABC

Nnodes: int

abstract getEleInteralForce()

ndf: int

node: list

nodeID0: float

nodeIDEnd: int

2.2 OpenSeesAnalyzer2D

class planesections.analysis.openSees.**OpenSeesAnalyzer2D**(*beam2D: ~planesections.builder.Beam*, *recorder=<class 'planesections.analysis.openSees.OutputRecorderOpenSees'>*, *geomTransform='Linear'*, *clearOld=True*)

Bases: object

This class is used to can be used to create and run an analysis of an input 2D beam using OpenSeesPy. The nodes, elements, sections, and forces for the beam are defined in the analysis model

Note, nodes and elements will both start at 0 instead of 1.

Parameters

- **beam** (*planesections Beam2D*) – The beam whose data is being recorded.
- **recorder** (*planesections Recorder*) – The recorder to use for the output beam.
- **geomTransform** (*str, optional*) – The OpenSees Geometry transform to use. Can be "Linear" or "PDelta"
- **clearOld** (*bool, optional*) – A flag that can be used to turn on or off clearing the old analysis when the beam is created. There are some very niche cases where users may want to have mutiple beams at once in the OpenSees model. However, this should remain true for nearly all analyses. Do not turn on unless you know what you're doing.

analyze()

Analyzes the model once and records outputs.

buildAnalysisPropreties()

Typical openSeesPy propreties that should work for any linear beam. A linear algorithm is used because there is no nonlienarity in the beam.

buildEleLoads()

Applies element loads to the appropriate elements in the model.

buildEulerBeams()

Creates an elastic Euler beam between each node in the model.

buildNodes()

Adds each node in the beam to the OpenSeesPy model, and assigns that node a fixity.

buildPointLoads()

Applies point loads to the appropriate nodes in the model.

initModel(*clearOld=True*)

Initializes the model.

Parameters

clearOld (*bool, optional*) – A flag that can be used to turn on or off clearing the old analysis when the beam is created. There are some very niche cases where users may want to have mutiple beams at once in the OpenSees model. However, this should remain true for nearly all analyses. Do not turn on unless you know what you're doing.

runAnalysis(*recordOutput=True*)

Makes and analyzes the beam in OpenSees.

Return type

None.

2.3 OpenSeesAnalyzer3D

class planesections.analysis.openSees.**OpenSeesAnalyzer3D**(*beam3D: ~planesections.builder.Beam*, *recorder=<class 'planesections.analysis.openSees.OutputRecorderOpenSees'>*, *geomTransform='Linear'*, *clearOld=True*)

Bases: object

This class is used to can be used to create and run an analysis of an input 2D beam using OpenSeesPy. The nodes, elements, sections, and forces for the beam are defined in the analysis model

Note, nodes and elements will both start at 0 instead of 1.

Parameters

- **beam** (*planesections Beam2D*) – The beam whose data is being recorded.
- **recorder** (*planesections Recorder*) – The recorder to use for the output beam.
- **geomTransform** (*str, optional*) – The OpenSees Geometry transform to use. Can be "Linear" or "PDelta"
- **clearOld** (*bool, optional*) – A flag that can be used to turn on or off clearing the old analysis when the beam is created. There are some very niche cases where users may want to have mutiple beams at once in the OpenSees model. However, this should remain true for nearly all analyses. Do not turn on unless you know what you're doing.

analyze()

Analyzes the model once and records outputs.

buildAnalysisPropreties()

Typical openSeesPy propreties that should work for any linear beam. A linear algorithm is used because there is no nonlienarity in the beam.

buildEleLoads()

Applies element loads to the appropriate elements in the model.

buildEulerBeams()

Creates an elastic Euler beam between each node in the model.

buildNodes()

Adds each node in the beam to the OpenSeesPy model, and assigns that node a fixity.

buildPointLoads()

Applies point loads to the appropriate nodes in the model.

initModel(*clearOld=True*)

Initializes the model.

Parameters

clearOld (*bool, optional*) – A flag that can be used to turn on or off clearing the old analysis when the beam is created. There are some very niche cases where users may want to have mutiple beams at once in the OpenSees model. However, this should remain true for nearly all analyses. Do not turn on unless you know what you're doing.

runAnalysis(*recordOutput=True*)

Makes and analyzes the beam in OpenSees.

Return type

None.

2.4 PyNiteAnalyzer2D

class planesections.analysis.pynite.**PyNiteAnalyzer2D**(*beam2D: ~planesections.builder.Beam*, *recorder=<class 'planesections.analysis.pynite.OutputRecorderPyNite2D'>*)

Bases: `object`

This class is used to can be used to create and run an analysis of an input 2D beam using OpenSeesPy. The nodes, elements, sections, and forces for the beam are defined in the analysis model

The PyNite solver makes use of a beam object, which is constructed and stored as a analysisBeam attribute

Note, nodes and elements will both start at 0 instead of 1.

For the PyNite beam, The 2D directions are X/Y

Parameters

- **beam** (*planesections Beam2D*) – The beam whose data is being recorded.
- **recorder** (*planesections Recorder*) – The recorder to use for the output beam.
- **geomTransform** (*str, optional*) – The OpenSees Geometry transform to use. Can be "Linear" or "PDelta"
- **clearOld** (*bool, optional*) – A flag that can be used to turn on or off clearing the old analysis when the beam is created. There are some very niche cases where users may want to have mutiple beams at once in the OpenSees model. However, this should remain true for nearly all analyses. Do not turn on unless you know what you're doing.

analyze()

Analyzes the model once and records outputs.

buildEleLoads()

Applies element loads to the appropriate elements in the model.

buildEulerBeams()

Creates an elastic Euler beam between each node in the model.

buildNodes()

Adds each node in the beam to the OpenSeesPy model, and assigns that node a fixity.

buildPointLoads()

Applies point loads to the appropriate nodes in the model.

initModel()

Initializes the model.

Parameters

clearOld (*bool, optional*) – A flag that can be used to turn on or off clearing the old analysis when the beam is created. There are some very niche cases where users may want to have mutiple beams at once in the OpenSees model. However, this should remain true for nearly all analyses. Do not turn on unless you know what you're doing.

runAnalysis(*recordOutput=True*)

Makes and analyzes the beam with PyNite.

Return type

None.

CHAPTER

THREE

DIAGRAM

The diagram module is used to plot representations of the beam. These output plots will include labels, loads, and support fixities. While the beam can analyze all support types, currently not all support types can be plotted. Only free, roller, pin, and fixed supports can be plotted. Similarly, the diagram only supports the plotting of vertical point loads, distributed loads, and moments.

It's recommended that the diagram features are accessed throught the beam plotter

1. *Plot Beam Diagram*
2. *Beam Plotter*

3.1 Plot Beam Diagram

The class used to plot beam diagrams. While the beam can analyze all support types, currently not all support types can be plotted. Only free, roller, pin, and fixed supports can be plotted.

3.2 Beam Plotter

The class used to plot beam diagrams.

It's highly recommened that plotBeamDiagram is used instead of BeamPlotter2D. While the beam can analyze all support types, currently not all support types can be plotted. Only free, roller, pin, and fixed supports can be plotted.

CHAPTER

FOUR

ENVIRONMENT

The Environment classes are used to manage global settings. These are used to modifyunits in beam diagrams, where the user can specify if they want imperial or metric unit labels. The user can also modify the number of decimals they want and other settings.

1. environment-diagramUnitEnvironmentHandler

CHAPTER

FIVE

POST-PROCESS

These classes are used to plot beam outputs.

1. *Get Beam Displacement*
2. *Get Beam Vertical Displacement*
3. *Get Beam Maximum Vertical Displacement*
4. *Plot Internal Forces*
5. *Plot Shear Force*
6. *Plot Moment*
7. *Plot Displacement*
8. *Plot Vertical Displacement*
9. *Plot Vertical Displacement*

5.1 Get Beam Displacement

planesections.postprocess.parse.**getDisp**(*beam:* Beam, *ind: int*)

Gets the beam displacement along the axis specified for the index.

Parameters

- **beam** (Beam) – The beam to read displacement from. The beam must be analyzed to get data.
- **ind** (*int*) – The index of the axis to read from. Can have value 0: horizontal displacement 1: vertical displacement 2: rotation.

Returns

- **disp** (*numpy array*) – The displacement at each x coordinant.
- **xcoords** (*numpy array*) – The x coordinants.

5.2 Get Beam Vertical Displacement

planesections.postprocess.parse.**getVertDisp**(*beam:* Beam)

Gets the beam vertical displacement for the beam

Parameters

beam (Beam) – The beam to read displacement from. The beam must be analyzed to get data.

Returns

- **disp** (*numpy array*) – The displacement at each x coordinant.
- **xcoords** (*numpy array*) – The x coordinants.

5.3 Get Beam Maximum Vertical Displacement

planesections.postprocess.parse.**getMaxVertDisp**(*beam:* Beam)

Gets the absolute value of beam vertical displacement and it's location.

Parameters

beam (Beam) – The beam to read displacement from. The beam must be analyzed to get data.

Returns

- **dispMax** (*float*) – The displacement at each x coordinant.
- **xcoords** (*numpy array*) – The x coordinants.

5.4 Plot Internal Forces

5.5 Plot Shear Force

5.6 Plot Moment

5.7 Plot Displacement

5.8 Plot Vertical Displacement

5.9 Plot Vertical Displacement

CHAPTER

SIX

DEVELOPED BY

Christian Slotboom https://github.com/cslotboom/planesections.

M.A.Sc. Structural Engineering
Engineer in Training

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

F

G

I

J

M

N

O

P

R

S

T

U

V

W