OpenSeesPyAssistant

Generated by Doxygen 1.9.3

Author: Carmine Schipani

Date: January 21 2022

Chapter 1

OpenSeesPyAssistant

Help the use of OpenSeesPy tools and the implementation of material models, elements, fibers and much more.

Chapter 2

Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 AnalysisAndPostProcessing Namespace Reference

Module with pre-made analysis and postprocessing functions.

Classes

- class Analysis

  Class dedicated to the analysis of the OpenSeesPy model.

6.1.1 Detailed Description

Module with pre-made analysis and postprocessing functions.

Carmine Schipani, 2021

6.2 Connections Namespace Reference

Module with different functions useful when defining boundary conditions (fix) or connections (pin, rigid or springs).

Functions

- def Pin (int NodeRID, int NodeCID)

  Function that constrains the translational DOF with a multi-point constraint.
- def RigidSupport (int NodeID)

  Function that fixes the x, y movements and the rotation of one node.
- def RotationalSpring (int ElementID, int NodeRID, int NodeCID, int MatID, Rigid=False)

  Function that defines a zero-length spring and constrains the translations DOFs of the spring.

6.2.1 Detailed Description

Module with different functions useful when defining boundary conditions (fix) or connections (pin, rigid or springs).

Carmine Schipani, 2021

6.2.2 Function Documentation

6.2.2.1 Pin()

```
def Connections.Pin (
            int NodeRID,
            int NodeCID )
```

Function that constrains the translational DOF with a multi-point constraint.

Parameters

NodeRID	(int): Node ID which will be retained by the multi-point constraint
NodeCID	(int): Node ID which will be constrained by the multi-point constraint

Exceptions

WrongArgument	The IDs passed needs to be different.
NegativeValue	The ID of NodeRID needs to be a positive integer.
NegativeValue	The ID of NodeCID needs to be a positive integer.

Definition at line 22 of file Connections.py.

```
00022 def Pin(NodeRID: int, NodeCID: int):
00023     """
00024     Function that constrains the translational DOF with a multi-point constraint.
00025 
00026     @param NodeRID (int): Node ID which will be retained by the multi-point constraint
00027     @param NodeCID (int): Node ID which will be constrained by the multi-point constraint
00028 
00029     @exception WrongArgument: The IDs passed needs to be different.
00030     @exception NegativeValue: The ID of NodeRID needs to be a positive integer.
00031     @exception NegativeValue: The ID of NodeCID needs to be a positive integer.
00032     """
00033     if NodeCID == NodeRID: raise WrongArgument()
00034     if NodeRID < 1: raise NegativeValue()
00035     if NodeCID < 1: raise NegativeValue()
00036 
00037     # Constrain the translational DOF with a multi-point constraint
00038     #           retained constrained DOF_1 DOF_2
00039     equalDOF(NodeRID, NodeCID, 1, 2)
00040 
00041 
```

6.2.2.2 RigidSupport()

```
def Connections.RigidSupport (
            int NodeID )
```

Function that fixes the x, y movements and the rotation of one node.

Parameters

NodeID	(int): ID of the node to be fixed

Exceptions

NegativeValue	The ID of NodeID needs to be a positive integer.

Definition at line 9 of file Connections.py.

```
00009 def RigidSupport(NodeID: int):
00010     """
00011     Function that fixes the x, y movements and the rotation of one node.
00012
00013     @param NodeID (int): ID of the node to be fixed
00014
00015     @exception NegativeValue: The ID of NodeID needs to be a positive integer.
00016     """
00017     if NodeID < 1: raise NegativeValue()
00018
00019     fix(NodeID, 1, 1, 1)
00020
00021
```

6.2.2.3 RotationalSpring()

```
def Connections.RotationalSpring (
            int ElementID,
            int NodeRID,
            int NodeCID,
            int MatID,
             Rigid = False )
```

Function that defines a zero-length spring and constrains the translations DOFs of the spring.

Can be used also to create rigid connections.

Parameters

ElementID	(int): ID of the zerolength element that models the spring
NodeRID	(int): Node ID which will be retained by the multi-point constraint
NodeCID	(int): Node ID which will be constrained by the multi-point constraint
MatID	(int): ID of the material model chosen
Rigid	(bool, optional): Optional argument that transforms the joint in a completely rigid connection. Defaults to False.

Exceptions

NegativeValue	The ID of ElementID needs to be a positive integer.
NegativeValue	The ID of NodeCID needs to be a positive integer.
NegativeValue	The ID of NodeRID needs to be a positive integer.
WrongArgument	The IDs of the nodes passed needs to be different.

Exceptions

NegativeValue	The ID of MatID needs to be a positive integer.

Definition at line 42 of file Connections.py.

```
00042 def RotationalSpring(ElementID: int, NodeRID: int, NodeCID: int, MatID: int, Rigid = False):
00043     """
00044     Function that defines a zero-length spring and constrains the translations DOFs of the spring. Can be used also to create rigid connections.
00045 
00046     @param ElementID (int): ID of the zerolength element that models the spring
00047     @param NodeRID (int): Node ID which will be retained by the multi-point constraint
00048     @param NodeCID (int): Node ID which will be constrained by the multi-point constraint
00049     @param MatID (int): ID of the material model chosen
00050     @param Rigid (bool, optional): Optional argument that transforms the joint in a completely rigid connection. Defaults to False.
00051 
00052     @exception NegativeValue: The ID of ElementID needs to be a positive integer.
00053     @exception NegativeValue: The ID of NodeCID needs to be a positive integer.
00054     @exception NegativeValue: The ID of NodeRID needs to be a positive integer.
00055     @exception WrongArgument: The IDs of the nodes passed needs to be different.
00056     @exception NegativeValue: The ID of MatID needs to be a positive integer.
00057     """
00058     if ElementID < 1: raise NegativeValue()
00059     if NodeCID < 1: raise NegativeValue()
00060     if NodeRID < 1: raise NegativeValue()
00061     if NodeCID == NodeRID: raise WrongArgument()
00062     if MatID < 1: raise NegativeValue()
00063 
00064     if not Rigid:
00065         # Zero length element (spring)
00066         element("zeroLength", ElementID, NodeRID, NodeCID, "-mat", MatID, "-dir", 6)
00067 
00068         # Constrain the translational DOF with a multi-point constraint
00069         Pin(NodeRID, NodeCID)
00070     else:
00071         equalDOF(NodeRID, NodeCID, 1, 2, 3)
00072 
00073 
00074 
00075 
```

6.3 Constants Namespace Reference

Module with the values of a set of essential constants.

Variables

- float G_CONST = 9.810∗m_unit/s_unit∗∗2
- int MAX_ITER = 100
- int MAX_ITER_INTEGRATION = 50
- float RIGID = 100.0
- float TOL = 1.0e-6
- float TOL_INTEGRATION = 1.0e-12
- float ZERO = 1.0e-9

6.3.1 Detailed Description

Module with the values of a set of essential constants.

They are consistent with the units defined in Units.
Carmine Schipani, 2021

6.3.2 Variable Documentation

6.3.2.1 G_CONST

```
float G_CONST = 9.810*m_unit/s_unit**2
```

Definition at line 11 of file Constants.py.

6.3.2.2 MAX_ITER

```
int MAX_ITER = 100
```

Definition at line 13 of file Constants.py.

6.3.2.3 MAX_ITER_INTEGRATION

```
int MAX_ITER_INTEGRATION = 50
```

Definition at line 14 of file Constants.py.

6.3.2.4 RIGID

```
float RIGID = 100.0
```

Definition at line 12 of file Constants.py.

6.3.2.5 TOL

```
float TOL = 1.0e-6
```

Definition at line 8 of file Constants.py.

6.3.2.6 TOL_INTEGRATION

```
float TOL_INTEGRATION = 1.0e-12
```

Definition at line 9 of file Constants.py.

6.3.2.7 ZERO

```
float ZERO = 1.0e-9
```

Definition at line 10 of file Constants.py.

6.4 DataManagement Namespace Reference

Module with the parent abstract class DataManagement.

Classes

- class DataManagement
  Abstract parent class for data management.

6.4.1 Detailed Description

Module with the parent abstract class DataManagement.

Carmine Schipani, 2021

6.5 ErrorHandling Namespace Reference

Module dedicated to the error handling.

Classes

- class InconsistentGeometry

    Exception class for the "inconsistent geometry" error.
- class MemberFailure

    Exception class for the "member failure" error.
- class NegativeValue

    Exception class for the "negative value (argument or result)" error.
- class NoApplicability

    Exception class for the "no applicability of formula of theory" error.
- class PositiveValue

    Exception class for the "positive value (argument or result)" error.
- class WrongArgument

    Exception class for the "input of a wrong argument" error.
- class WrongDimension

    Exception class for the "wrong array dimensions" error.
- class WrongNodeIDConvention

    Exception class for the "wrong node ID convention definition" error.
- class ZeroDivision

    Exception class for the "zero division" error.
- class ZeroLength

    Exception class for the "zero length element (non intentional)" error.

6.5.1 Detailed Description

Module dedicated to the error handling.

Carmine Schipani, 2021

6.6 Fibers Namespace Reference

Module for the fibers (rectangular, circular and I shape).

Classes

- class Fibers

    Parent abstract class for the storage and manipulation of a fiber's information (mechanical and geometrical parameters, etc) and initialisation in the model.
- class FibersCirc

    Class that stores funcions, material properties, geometric and mechanical parameters for a circular RC fiber section.
- class FibersCircRCCircShape

    Class that is the children of FibersCirc and combine the class RCCircShape (section) to retrieve the information needed.
- class FibersIShape

    Class that stores funcions, material properties, geometric and mechanical parameters for a steel I shape (non double symmetric) fiber section.
- class FibersIShapeSteelIShape

Class that is the children of FibersIShape and combine the class SteelIShape (section) to retrieve the information needed.

- class FibersRect

    Class that stores funcions, material properties, geometric and mechanical parameters for a rectangular RC fiber section.

- class FibersRectRCRectShape

    Class that is the children of FibersRect and combine the class RCRectShape (section) to retrieve the information needed.

Functions

- def create_fiber_section (fiber_info)

    Initialise fiber cross-section with OpenSeesPy commands.

- def plot_fiber_section (fiber_info, fill_shapes=True, matcolor=['#808080', '#D3D3D3', 'r', 'b', 'g', 'y'])

    Plot fiber cross-section.

6.6.1 Detailed Description

Module for the fibers (rectangular, circular and I shape).

Carmine Schipani, 2021

6.6.2 Function Documentation

6.6.2.1 create_fiber_section()

```
def Fibers.create_fiber_section (
            fiber_info )
```

Initialise fiber cross-section with OpenSeesPy commands.

For examples, see plot_fiber_section. Inspired by fib_sec_list_to_cmds from ops_vis written by Seweryn Kokot

Parameters

fiber_info	(list): List of lists (be careful with the local coordinate system!). The first list defines the fiber section: ['section', 'Fiber', ID, '-GJ', GJ] The other lists have one of the following format (coordinate input: (y, z)!): ['layer', 'bar', mat_ID, A, y, z] # one bar ['layer', 'straight', mat_ID, n_bars, A, yI, zI, yJ, zJ] # line range of bars (with I = first bar, J = last bar) ['layer', 'circ', mat_ID, n_bars, A, yC, zC, r, (a0_deg), (a1_deg)] # circular range of bars (with C = center, r = radius) ['patch', 'rect', mat_ID, ∗discr, -yI, zI, yK, -zK] # rectangle (with yI = yK = d/2; zI = zK = b/2) ['patch', 'quad', mat_ID, ∗discr, yI, zI, yJ, zJ, yK, zK, yL, zL] # quadrilateral shaped (starting from bottom left, counterclockwise: I, J, K, L) ['patch', 'circ', mat_ID, ∗discr, yC, zC, ri, re, (a0), (a1)] # (with C = center, ri = internal radius, re = external radius)

Definition at line 857 of file Fibers.py.

```
00857 def create_fiber_section(fiber_info):
00858     """
00859     Initialise fiber cross-section with OpenSeesPy commands.
00860     For examples, see plot_fiber_section.
00861     Inspired by fib_sec_list_to_cmds from ops_vis written by Seweryn Kokot
00862 
00863     @param fiber_info (list): List of lists (be careful with the local coordinate system!). The first list defines the fiber section: \n
00864         ['section', 'Fiber', ID, '-GJ', GJ] \n
00865         The other lists have one of the following format (coordinate input: (y, z)!): \n
00866         ['layer', 'bar', mat_ID, A, y, z] # one bar \n
00867         ['layer', 'straight', mat_ID, n_bars, A, yI, zI, yJ, zJ] # line range of bars (with I = first bar, J = last bar) \n
00868         ['layer', 'circ', mat_ID, n_bars, A, yC, zC, r, (a0_deg), (a1_deg)] # circular range of bars (with C = center, r = radius) \n
00869         ['patch', 'rect', mat_ID, *discr, -yI, zI, yK, -zK] # rectangle (with yI = yK = d/2; zI = zK = b/2) \n
00870         ['patch', 'quad', mat_ID, *discr, yI, zI, yJ, zJ, yK, zK, yL, zL] # quadrilateral shaped (starting from bottom left, counterclockwise: I, J, K, L) \n
00871         ['patch', 'circ', mat_ID, *discr, yC, zC, ri, re, (a0), (a1)] #  (with C = center, ri = internal radius, re = external radius)
00872     """
00873     for dat in fiber_info:
00874         if dat[0] == 'section':
00875             fib_ID, GJ = dat[2], dat[4]
00876             section('Fiber', fib_ID, 'GJ', GJ)
00877 
00878         if dat[0] == 'layer':
00879             mat_ID = dat[2]
00880             if dat[1] == 'straight':
00881                 n_bars = dat[3]
00882                 As = dat[4]
00883                 Iy, Iz, Jy, Jz = dat[5], dat[6], dat[7], dat[8]
00884                 layer('straight', mat_ID, n_bars, As, Iy, Iz, Jy, Jz)
00885             if dat[1] == 'bar':
00886                 As = dat[3]
00887                 Iy = dat[4]
00888                 Iz = dat[5]
00889                 fiber(Iy, Iz, As, mat_ID)
00890                 # layer('straight', mat_ID, 1, As, Iy, Iz, Iy, Iz)
00891             if dat[1] == 'circ':
00892                 n_bars, As = dat[3], dat[4]
00893                 yC, zC, r = dat[5], dat[6], dat[7]
00894                 if len(dat) > 8:
00895                     a0_deg = dat[8]
00896                 else:
00897                     a0_deg = 0.
00898                 a1_deg = 360. - 360./n_bars + a0_deg
00899                 if len(dat) > 9: a1_deg = dat[9]
00900                 layer('circ', mat_ID, n_bars, As, yC, zC, r, a0_deg, a1_deg)
00901 
00902         if dat[0] == 'patch':
00903             mat_ID = dat[2]
00904             nIJ = dat[4]
00905             nJK = dat[3]
00906 
00907             if dat[1] == 'quad' or dat[1] == 'quadr':
00908                 Iy, Iz, Jy, Jz = dat[5], dat[6], dat[7], dat[8]
00909                 Ky, Kz, Ly, Lz = dat[9], dat[10], dat[11], dat[12]
00910                 patch('quad', mat_ID, nIJ, nJK, Iy, Iz, Jy, Jz, Ky, Kz,
00911                         Ly, Lz)
00912 
00913             if dat[1] == 'rect':
00914                 Iy, Iz, Ky, Kz = dat[5], dat[6], dat[7], dat[8]
00915                 patch('rect', mat_ID, nIJ, nJK, Iy, Iz, Ky, Kz)
00916                 # patch('rect', mat_ID, nIJ, nJK, Iy, Kz, Ky, Iz)
00917 
00918             if dat[1] == 'circ':
00919                 mat_ID, nc, nr = dat[2], dat[3], dat[4]
00920                 yC, zC, ri, re = dat[5], dat[6], dat[7], dat[8]
00921                 if len(dat) > 9:
00922                     a0 = dat[9]
00923                 else:
00924                     a0 = 0.
00925                 a1 = 360. + a0
00926                 if len(dat) > 10: a1 = dat[10]
00927                 patch('circ', mat_ID, nc, nr, yC, zC, ri, re, a0, a1)
00928 
```

6.6.2.2 plot_fiber_section()

```
def Fibers.plot_fiber_section (
            fiber_info,
            fill_shapes = True,
            matcolor = ['#808080', '#D3D3D3', 'r', 'b', 'g', 'y'] )
```

Plot fiber cross-section.

Coordinate system used: plotting coordinte = (x, y), fiber section coordinate (z, y) = (-x, y)
Inspired by plot_fiber_section from ops_vis written by Seweryn Kokot.

Parameters

fiber_info	(list): List of lists (be careful with the local coordinate system!). The first list defines the fiber section: ['section', 'Fiber', ID, '-GJ', GJ] The other lists have one of the following format (coordinate input: (y, z)!): ['layer', 'bar', mat_ID, A, y, z] # one bar ['layer', 'straight', mat_ID, n_bars, A, yI, zI, yJ, zJ] # line range of bars (with I = first bar, J = last bar) ['layer', 'circ', mat_ID, n_bars, A, yC, zC, r, (a0_deg), (a1_deg)] # circular range of bars (with C = center, r = radius) ['patch', 'rect', mat_ID, ∗discr, -yI, zI, yK, -zK] # rectangle (with yI = yK = d/2; zI = zK = b/2) ['patch', 'quad', mat_ID, ∗discr, yI, zI, yJ, zJ, yK, zK, yL, zL] # quadrilateral shaped (starting from bottom left, counterclockwise: I, J, K, L) ['patch', 'circ', mat_ID, ∗discr, yC, zC, ri, re, (a0), (a1)] # (with C = center, ri = internal radius, re = external radius)
fill_shapes	(bool, optional): Option to fill fibers with color specified in matcolor. Defaults to True.
matcolor	(list, optional): List of colors for various material IDs. Defaults to ['#808080', '#D3D3D3', 'r', 'b', 'g', 'y'].

Example 1: Simple rectangle with 2 rebars (D = diameter) on top (e distance from the top and from the lateral borders). Rectangle with first corner = I (bottom right) and second corner = K (top left); number of fibers = discr (list of 2) fib_sec = [['section', 'Fiber', ID, '-GJ', GJ], ['patch', 'rect', concrete_mat_ID, ∗discr, yI, zI, yK, zK], ['layer', 'bar', bars_mat_ID, Ay, yI-e-D/2, zI-e-D/2], # left rebar ['layer', 'bar', bars_mat_ID, Ay, yI-e-D/2, -(zI-e-D/2)]] # right rebar

Example 2: double symmetric I shape. Each rectangle (2 flanges and 1 web): first corner = I (bottom right) and second corner = K (top left); number of fibers = discr (list of 2) fib_sec = [['section', 'Fiber', ID, '-GJ', GJ], ['patch', 'rect', mat_ID, ∗discr, yI_tf, zI_tf, yK_tf, zK_tf], # top flange ['patch', 'rect', mat_ID, ∗discr, yI_bf, zI_bf, yK_bf, zK_bf], # bottom flange ['patch', 'rect', mat_ID, ∗discr, yI_w, zI_w, yK_w, zK_w]] # web

Definition at line 673 of file Fibers.py.

```
00673 def plot_fiber_section(fiber_info, fill_shapes = True, matcolor=['#808080', '#D3D3D3', 'r', 'b', 'g',
      'y']):
00674     """
00675     Plot fiber cross-section. Coordinate system used: plotting coordinte = (x, y), fiber section
      coordinate (z, y) = (-x, y)
00676     Inspired by plot_fiber_section from ops_vis written by Seweryn Kokot.
00677
00678     @param fiber_info (list): List of lists (be careful with the local coordinate system!). The first
      list defines the fiber section: \n
00679         ['section', 'Fiber', ID, '-GJ', GJ] \n
00680         The other lists have one of the following format (coordinate input: (y, z)!): \n
00681         ['layer', 'bar', mat_ID, A, y, z] # one bar \n
00682         ['layer', 'straight', mat_ID, n_bars, A, yI, zI, yJ, zJ] # line range of bars (with I = first
      bar, J = last bar) \n
00683         ['layer', 'circ', mat_ID, n_bars, A, yC, zC, r, (a0_deg), (a1_deg)] # circular range of bars
      (with C = center, r = radius) \n
00684         ['patch', 'rect', mat_ID, *discr, -yI, zI, yK, -zK] # rectangle (with yI = yK = d/2; zI = zK =
      b/2) \n
00685         ['patch', 'quad', mat_ID, *discr, yI, zI, yJ, zJ, yK, zK, yL, zL] # quadrilateral shaped
      (starting from bottom left, counterclockwise: I, J, K, L) \n
```

```
        ['patch', 'circ', mat_ID, *discr, yC, zC, ri, re, (a0), (a1)] #  (with C = center, ri = internal radius, re = external radius)
    @param fill_shapes (bool, optional): Option to fill fibers with color specified in matcolor. Defaults to True.
    @param matcolor (list, optional): List of colors for various material IDs. Defaults to ['#808080', '#D3D3D3', 'r', 'b', 'g', 'y'].

    Example 1: Simple rectangle with 2 rebars (D = diameter) on top (e distance from the top and from the lateral borders).
        Rectangle with first corner =  I (bottom right) and second corner = K (top left); number of fibers = discr (list of 2)
        fib_sec = [['section', 'Fiber', ID, '-GJ', GJ],
            ['patch', 'rect', concrete_mat_ID, *discr, yI, zI, yK, zK],
            ['layer', 'bar', bars_mat_ID, Ay, yI-e-D/2, zI-e-D/2], # left rebar
            ['layer', 'bar', bars_mat_ID, Ay, yI-e-D/2, -(zI-e-D/2)]] # right rebar

    Example 2: double symmetric I shape.
        Each rectangle (2 flanges and 1 web): first corner =  I (bottom right) and second corner = K (top left); number of fibers = discr (list of 2)
        fib_sec = [['section', 'Fiber', ID, '-GJ', GJ],
            ['patch', 'rect', mat_ID, *discr, yI_tf, zI_tf, yK_tf, zK_tf], # top flange
            ['patch', 'rect', mat_ID, *discr, yI_bf, zI_bf, yK_bf, zK_bf], # bottom flange
            ['patch', 'rect', mat_ID, *discr, yI_w, zI_w, yK_w, zK_w]] # web
    """

    mat_to_col = {}
    fig, ax = plt.subplots()
    ax.grid(False)

    for item in fiber_info:
        if item[0] == 'section':
            fib_ID = item[2]
        if item[0] == 'layer':
            matID = item[2]
            mat_to_col = __assignColorToMat(matID, mat_to_col, matcolor)
            if item[1] == 'bar':
                As = item[3]
                Iy = item[4]
                Iz = item[5]
                r = np.sqrt(As / np.pi)
                bar = Circle((-Iz, Iy), r, ec='k', fc='k', zorder=10)
                ax.add_patch(bar)
            if item[1] == 'straight':
                n_bars = item[3]
                As = item[4]
                Iy, Iz, Jy, Jz = item[5], item[6], item[7], item[8]
                r = np.sqrt(As / np.pi)
                Y = np.linspace(Iy, Jy, n_bars)
                Z = np.linspace(Iz, Jz, n_bars)
                for zi, yi in zip(Z, Y):
                    bar = Circle((-zi, yi), r, ec='k', fc=mat_to_col[matID], zorder=10)
                    ax.add_patch(bar)
            if item[1] == 'circ':
                n_bars, As = item[3], item[4]
                yC, zC, r = item[5], item[6], item[7]
                if len(item) > 8:
                    a0_deg = item[8]
                else:
                    a0_deg = 0.
                a1_deg = 360. - 360./n_bars + a0_deg
                if len(item) > 9: a1_deg = item[9]

                a0_rad, a1_rad = np.pi * a0_deg / 180., np.pi * a1_deg / 180.
                r_bar = np.sqrt(As / np.pi)
                thetas = np.linspace(a0_rad, a1_rad, n_bars)
                Y = yC + r * np.cos(thetas)
                Z = zC + r * np.sin(thetas)
                for zi, yi in zip(Z, Y):
                    bar = Circle((-zi, yi), r_bar, ec='k', fc=mat_to_col[matID], zorder=10)
                    ax.add_patch(bar)

        if (item[0] == 'patch' and (item[1] == 'quad' or item[1] == 'quadr' or
                                  item[1] == 'rect')):
            matID, nIJ, nJK = item[2], item[3], item[4]
            mat_to_col = __assignColorToMat(matID, mat_to_col, matcolor)


            if item[1] == 'quad' or item[1] == 'quadr':
                Iy, Iz, Jy, Jz = item[5], item[6], item[7], item[8]
                Ky, Kz, Ly, Lz = item[9], item[10], item[11], item[12]

            if item[1] == 'rect':
                Iy, Iz, Ky, Kz = item[5], item[6], item[7], item[8]
                Jy, Jz, Ly, Lz = Iy, Kz, Ky, Iz
                # check order of definition
                if Kz-Iz < 0 or Ky-Iy < 0: print("!!!!!!! WARNING !!!!!!! The fiber is not defined bottom right, top left")
```

```

            # check for convexity (vector products)
            outIJxIK = (Jy-Iy)*(Kz-Iz) - (Ky-Iy)*(Jz-Iz)
            outIKxIL = (Ky-Iy)*(Lz-Iz) - (Ly-Iy)*(Kz-Iz)
            # check if I, J, L points are colinear
            outIJxIL = (Jy-Iy)*(Lz-Iz) - (Ly-Iy)*(Jz-Iz)
            # outJKxJL = (Ky-Jy)*(Lz-Jz) - (Ly-Jy)*(Kz-Jz)

            if -outIJxIK <= 0 or -outIKxIL <= 0 or -outIJxIL <= 0:
                print('!!!!!!! WARNING !!!!!!! Patch quad is non-convex or non-counter-clockwise defined or has at least 3 colinear points in line')

            IJz, IJy = np.linspace(Iz, Jz, nIJ+1), np.linspace(Iy, Jy, nIJ+1)
            JKz, JKy = np.linspace(Jz, Kz, nJK+1), np.linspace(Jy, Ky, nJK+1)
            LKz, LKy = np.linspace(Lz, Kz, nIJ+1), np.linspace(Ly, Ky, nIJ+1)
            ILz, ILy = np.linspace(Iz, Lz, nJK+1), np.linspace(Iy, Ly, nJK+1)

            if fill_shapes:
                Z = np.zeros((nIJ+1, nJK+1))
                Y = np.zeros((nIJ+1, nJK+1))

                for j in range(nIJ+1):
                    Z[j, :] = np.linspace(IJz[j], LKz[j], nJK+1)
                    Y[j, :] = np.linspace(IJy[j], LKy[j], nJK+1)

                for j in range(nIJ):
                    for k in range(nJK):
                        zy = np.array([[-Z[j, k], Y[j, k]],
                                       [-Z[j, k+1], Y[j, k+1]],
                                       [-Z[j+1, k+1], Y[j+1, k+1]],
                                       [-Z[j+1, k], Y[j+1, k]]])
                        poly = Polygon(zy, True, ec='k', fc=mat_to_col[matID])
                        ax.add_patch(poly)

            else:
                # horizontal lines
                for az, bz, ay, by in zip(IJz, LKz, IJy, LKy):
                    plt.plot([-az, -bz], [ay, by], 'b-', zorder=1)

                # vertical lines
                for az, bz, ay, by in zip(JKz, ILz, JKy, ILy):
                    plt.plot([-az, -bz], [ay, by], 'b-', zorder=1)

        if item[0] == 'patch' and item[1] == 'circ':
            matID, nc, nr = item[2], item[3], item[4]
            mat_to_col = __assignColorToMat(matID, mat_to_col, matcolor)

            yC, zC, ri, re = item[5], item[6], item[7], item[8]
            if len(item) > 9:
                a0 = item[9]
            else:
                a0 = 0.
            a1 = 360. + a0
            if len(item) > 10: a1 = item[10]

            dr = (re - ri) / nr
            dth = (a1 - a0) / nc

            for j in range(nr):
                rj = ri + j * dr
                rj1 = rj + dr

                for i in range(nc):
                    thi = a0 + i * dth
                    thi1 = thi + dth
                    wedge = Wedge((-zC, yC), rj1, thi, thi1, width=dr, ec='k', #Seweryn Kokot: (yC, -zC), wrong??
                        lw=1, fc=mat_to_col[matID])
                    ax.add_patch(wedge)
    ax.set(xlabel='x dimension [{}]'.format(length_unit), ylabel='y dimension [{}]'.format(length_unit),
                    title='Fiber section (ID = {})'.format(fib_ID))
    ax.axis('equal')


```

6.7 FunctionalFeatures Namespace Reference

Module with useful functions (discretise curves, ID conventions, etc)
Carmine Schipani, 2021.

Classes

- class IDGenerator

    Class that manage the ID generation.

Functions

- def DiscretizeLinearly (np.ndarray LP, int discr, plot=False, block=False, show_original_LP=True)

    This function discretize the curve 'LP' given adding the number of point given by 'discr' between every point (linearly).
- def DiscretizeLoadProtocol (np.ndarray SDR_LP, np.ndarray nr_cycles_LP, int discr_first_cycle, plot=False, block=False, show_original_peaks=True)

    Discretized a cyclic load protocol keeping a similar discretisation step throughout the different cycles and keeping in the output the extremes (peaks).
- def GridIDConvention (int pier_axis, int floor_axis, max_pier=-1, max_floor=-1)

    Function used to construct the ID of the nodes in the grid (first nodes that define the geometry of the model).
- def IDConvention (int prefix, int suffix, n_zeros_between=0)

    Function used to construct IDs for elements and offgrid nodes.
- def NodesOrientation (int iNode_ID, int jNode_ID)

    Function that finds the orientation of the vector with direction 'jNode_ID''iNode_ID'.
- def OffsetNodeIDConvention (int node_ID, str orientation, str position_i_or_j)

    Function used to add node on top of existing ones in the extremes of memebers with springs.
- def plot_member (list element_array, member_name="Member name not defined", show_element_ID=True, show_node_ID=True)

    Function that plots a set of elements.
- def plot_nodes (list nodes_array, name="Not defined", show_node_ID=True)

    Function that plots a set of nodes.
- def ProgressingPercentage (max_iter, int i, int next_step, step=10)

    Function that shows the progressing percentage of an iterative process.

6.7.1 Detailed Description

Module with useful functions (discretise curves, ID conventions, etc)
Carmine Schipani, 2021.

6.7.2 Function Documentation

6.7.2.1 DiscretizeLinearly()

```
def FunctionalFeatures.DiscretizeLinearly (
             np.ndarray LP,
             int discr,
              plot = False,
              block = False,
              show_original_LP = True )
```

This function discretize the curve 'LP' given adding the number of point given by 'discr' between every point (linearly).

Parameters

LP	(np.ndarray): Array (1 dimension) that stores the curve that needs to be discretized
discr	(int): The number of points to add between every two points of 'LP' (linearly)
plot	(bool, optional): Option to show the plot of the discretized (and also the original LP). Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
show_original_LP	(bool, optional): Option to show the original LP to check if the discretized curve is correct. Defaults to True.

Returns

np.ndarray: Array (1 dimension) that stores the new discretized load protocol.

Definition at line 99 of file FunctionalFeatures.py.

```
00099 def DiscretizeLinearly(LP: np.ndarray, discr: int, plot = False, block = False, show_original_LP = True):
00100     """
00101     This function discretize the curve 'LP' given adding the number of point given by 'discr' between every point (linearly).
00102
00103     @param LP (np.ndarray): Array (1 dimension) that stores the curve that needs to be discretized
00104     @param discr (int): The number of points to add between every two points of 'LP' (linearly)
00105     @param plot (bool, optional): Option to show the plot of the discretized (and also the original LP). Defaults to False.
00106     @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
00107     @param show_original_LP (bool, optional): Option to show the original LP to check if the discretized curve is correct. Defaults to True.
00108
00109     @returns np.ndarray: Array (1 dimension) that stores the new discretized load protocol.
00110     """
00111
00112     #TODO: check discr nonnegative int and LP 1 dimension
00113
00114     # Define the new discretized LP
00115     length = 1 + (np.size(LP)-1) * (discr+1)
00116     discr_LP = np.zeros(length)
00117
00118     # Performa manually the first iteration
00119     yprev = LP[0]
00120     x = [0, 1]
00121     discr_LP[0] = yprev
00122     iter = 0
00123
00124     # add the other points and the discretized ones
00125     for ynext in LP[1:]:
00126         y = [yprev, ynext]
00127
00128         # Compute new points
00129         xnew = np.linspace(x[0], x[1], discr+2)
00130         ynew = np.interp(xnew[1:], x, y)
00131
00132         # Add to the recording vector discr_LP
00133         index = np.array(np.arange(discr+1)+1+iter)
00134         discr_LP[index] = ynew
00135
00136         # Prepare for next iteration
00137         yprev = ynext
00138         iter = iter + discr + 1
00139
00140     if plot:
00141         fig, ax = plt.subplots()
00142         ax.plot(discr_LP, '-r', label='Discretised LP')
00143
00144         ax.set(xlabel='Step number [-]', ylabel='Unit of the loading protocol',
00145             title='Discretized loading protocol')
00146         ax.grid()
00147
00148         if show_original_LP:
00149             x_val = np.arange(0, np.size(discr_LP), discr+1)
00150             ax.plot(x_val, LP, 'ob', label='Original LP')
00151             ax.legend()
00152
00153         if block:
```

```
            plt.show()

    return discr_LP


```

6.7.2.2 DiscretizeLoadProtocol()

```
def FunctionalFeatures.DiscretizeLoadProtocol (
            np.ndarray SDR_LP,
            np.ndarray nr_cycles_LP,
            int discr_first_cycle,
             plot = False,
             block = False,
             show_original_peaks = True )
```

Discretized a cyclic load protocol keeping a similar discretisation step throughout the different cycles and keeping in the output the extremes (peaks).

Parameters

SDR_LP	(np.ndarray): Array (1 dimension) that stores the peaks of the cycles. They needs to be only the positive peaks, beacuse this function will use them as the extreme for each cycle.
nr_cycles_LP	(np.ndarray): Array (1 dimension) that stores the number of cycles for every extreme declared in 'SDR_LP' and its countepart negative. They need to be positive integers.
discr_first_cycle	(int): The number of points from peak to peak (counting the two peaks) in the first cycle. It should be odd.
plot	(bool, optional): Option to show the plot of the discretized (and also the original peaks). Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
show_original_peaks	(bool, optional): Option to show the original peaks to check if the discretized curve is correct. The argument plot need to be True. Defaults to True.

Exceptions

WrongDimension	SDR_LP and nr_cycles_LP need to be of same length.
NegativeValue	SDR_LP needs to have only positive integers.
NegativeValue	nr_cycles_LP needs to have only positive integers.
NegativeValue	discr_first_cycle needs to be a positive integer.

Returns

np.array: Array (1 dimension) that stores the new discretized load protocol curve.

Definition at line 32 of file FunctionalFeatures.py.

```
00032 def DiscretizeLoadProtocol(SDR_LP: np.ndarray, nr_cycles_LP: np.ndarray, discr_first_cycle: int, plot
      = False, block = False, show_original_peaks = True):
00033     """
00034     Discretized a cyclic load protocol keeping a similar discretisation step throughout the different
      cycles and keeping in the output the extremes (peaks).
00035
```

```
    @param SDR_LP (np.ndarray): Array (1 dimension) that stores the peaks of the cycles.
        They needs to be only the positive peaks, beacuse this function will use them as the extreme for each cycle.
    @param nr_cycles_LP (np.ndarray): Array (1 dimension) that stores the number of cycles for every extreme declared in 'SDR_LP' and its countepart negative.
        They need to be positive integers.
    @param discr_first_cycle (int): The number of points from peak to peak (counting the two peaks) in the first cycle. It should be odd.
    @param plot (bool, optional): Option to show the plot of the discretized (and also the original peaks). Defaults to False.
    @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
    @param show_original_peaks (bool, optional): Option to show the original peaks to check if the discretized curve is correct.
        The argument plot need to be True. Defaults to True.

    @exception WrongDimension: SDR_LP and nr_cycles_LP need to be of same length.
    @exception NegativeValue: SDR_LP needs to have only positive integers.
    @exception NegativeValue: nr_cycles_LP needs to have only positive integers.
    @exception NegativeValue: discr_first_cycle needs to be a positive integer.

    @returns np.array: Array (1 dimension) that stores the new discretized load protocol curve.
    """
    if np.size(SDR_LP) != np.size(nr_cycles_LP): raise WrongDimension()
    if any(col < 0 for col in SDR_LP): raise NegativeValue()
    if any(col < 0 for col in nr_cycles_LP): raise NegativeValue()
    if discr_first_cycle < 0: raise NegativeValue()

    if discr_first_cycle % 2 == 0:
            discr_first_cycle = discr_first_cycle + 1
    discr_factor = discr_first_cycle / (SDR_LP[0]*2)
    discretized_LP = [0.0]
    x_val = []
    skip_x = 0
    for i in range(np.size(SDR_LP)):
        discr_i = math.ceil(discr_factor*SDR_LP[i]*2)-1;
        if discr_i % 2 == 0:
            discr_i = discr_i + 1
        length_tmp = int((discr_i+1)/2)
        tmp_up = np.linspace(0.0, SDR_LP[i], length_tmp)
        tmp_down = np.linspace(SDR_LP[i], 0.0, length_tmp)
        for j in range(int(nr_cycles_LP[i])):
            discretized_LP = np.append(discretized_LP, tmp_up[1:length_tmp])
            discretized_LP = np.append(discretized_LP, tmp_down[1:length_tmp])
            discretized_LP = np.append(discretized_LP, -tmp_up[1:length_tmp])
            discretized_LP = np.append(discretized_LP, -tmp_down[1:length_tmp])
        # for check of original peaks
        x_val.append(length_tmp-1+skip_x)
        skip_x = (length_tmp-1)*(4*(nr_cycles_LP[i]-1)+3)+x_val[-1]


    if plot:
        fig, ax = plt.subplots()
        ax.plot(discretized_LP, '-r', label='Discretised LP')

        ax.set(xlabel='Step number [-]', ylabel='Unit of the loading protocol',
            title='Discretized loading protocol')
        ax.grid()

        if show_original_peaks:
            ax.plot(x_val, SDR_LP, 'ob', label='Original LP')
            ax.legend()

        if block:
            plt.show()

    return discretized_LP


```

6.7.2.3 GridIDConvention()

```
def FunctionalFeatures.GridIDConvention (
            int pier_axis,
            int floor_axis,
             max_pier = -1,
             max_floor = -1 )
```

Function used to construct the ID of the nodes in the grid (first nodes that define the geometry of the model).

The conventional grid node ID is xy, with x = the pier position 'pier_axis'; y = the floor position 'floor_axis'.

Parameters

pier_axis	(int): The pier (or x) postion of the node.
floor_axis	(int): The floor (or y) position of the node.
max_pier	(int, optional): Maximal pier position of the model (used to identify the number of digits). Defaults to -1, e.g. taken equal of pier_axis.
max_floor	(int, optional): Maximal floor position of the model (used to identify the number of digits). Defaults to -1, e.g. taken equal of floor_axis.

Exceptions

NameError	Work In Progress: only 9 floors/bays.
NegativeValue	The argument pier_axis needs to be a positive integer.
NegativeValue	The argument floor_axis needs to be a positive integer.
NegativeValue	The argument max_pier needs to be a positive integer if different from -1.
NegativeValue	The argument max_floor needs to be a positive integer if different from -1.
WrongArgument	The argument max_pier need to be equal or bigger to pier_axis
WrongArgument	The argument max_floor need to be equal or bigger to floor_axis

Returns

int: The grid node ID

Definition at line 159 of file FunctionalFeatures.py.

```
00159 def GridIDConvention(pier_axis: int, floor_axis: int, max_pier = -1, max_floor = -1):
00160     """
00161     Function used to construct the ID of the nodes in the grid (first nodes that define the geometry of the model).
00162     The conventional grid node ID is xy, with x = the pier position 'pier_axis'; y = the floor position 'floor_axis'.
00163 
00164     @param pier_axis (int): The pier (or x) postion of the node.
00165     @param floor_axis (int): The floor (or y) position of the node.
00166     @param max_pier (int, optional): Maximal pier position of the model (used to identify the number of digits).
00167         Defaults to -1, e.g. taken equal of pier_axis.
00168     @param max_floor (int, optional): Maximal floor position of the model (used to identify the number of digits).
00169         Defaults to -1, e.g. taken equal of floor_axis.
00170 
00171     @exception NameError: Work In Progress: only 9 floors/bays.
00172     @exception NegativeValue: The argument pier_axis needs to be a positive integer.
00173     @exception NegativeValue: The argument floor_axis needs to be a positive integer.
00174     @exception NegativeValue: The argument max_pier needs to be a positive integer if different from -1.
00175     @exception NegativeValue: The argument max_floor needs to be a positive integer if different from -1.
00176     @exception WrongArgument: The argument max_pier need to be equal or bigger to pier_axis
00177     @exception WrongArgument: The argument max_floor need to be equal or bigger to floor_axis
00178 
00179     @returns int: The grid node ID
00180     """
00181     # Convention:
00182     #       GridNodeID:     xy          with x = pier, y = floor
00183     if pier_axis > 9 or floor_axis > 9 or max_pier > 9 or max_floor > 9: raise NameError("WIP: maximal 9 floors or bays")
00184     max_pier = pier_axis if max_pier == -1 else max_pier
00185     max_floor = floor_axis if max_floor == -1 else max_floor
00186 
00187     if pier_axis < 0: raise NegativeValue()
00188     if floor_axis < 0: raise NegativeValue()
00189     if max_pier < 0: raise NegativeValue()
```

```
00190     if max_floor < 0: raise NegativeValue()
00191     if max_pier < pier_axis: raise WrongArgument()
00192     if max_floor < floor_axis: raise WrongArgument()
00193
00194     max_x_digits = int(math.log10(max_pier))+1
00195     max_y_digits = int(math.log10(max_floor))+1
00196
00197     # return 10**(max_x_digits+max_y_digits) + pier_axis*10**max_y_digits + floor_axis # with 1 as prefix (to consider more than one digit per axis, but exceed max ID)
00198     return pier_axis*10**max_y_digits + floor_axis
00199
00200
```

6.7.2.4 IDConvention()

```
def FunctionalFeatures.IDConvention (
            int prefix,
            int suffix,
             n_zeros_between = 0 )
```

Function used to construct IDs for elements and offgrid nodes.

It appends to a positive integer number 'prefix' a number of zeros 'n_zeros_between' and at last another positive integer 'suffix'. The conventional element ID is xy(a)x'y'(a') with xya = the node ID in pier x, floor y and offgrid parameter a (optional); x'y'a' = the node ID in pier x', floor y' and offgrid parameter a' (optional). For more information on x and y, see GridIDConvention; for more information on a, see OffsetNodeIDConvention.

Parameters

prefix	(int): Prefix of the new ID. For a vertical element it should be the left node ID; for an horizontal one it should be the bottom node.
suffix	(int): Suffix of the new ID. For a vertical element it should be the right node ID; for an horizontal one it should be the top node.
n_zeros_between	(int, optional): Number of zeros to add between the two nodes. Defaults to 0.

Exceptions

NegativeValue	The argument prefix needs to be a positive integer.
NegativeValue	The argument suffix needs to be a positive integer.
NegativeValue	The argument n_zeros_between needs to be a positive integer.

Returns

int: The combined ID

Definition at line 201 of file FunctionalFeatures.py.

```
00201 def IDConvention(prefix: int, suffix: int, n_zeros_between = 0):
00202     """
00203     Function used to construct IDs for elements and offgrid nodes.
00204     It appends to a positive integer number 'prefix' a number of zeros 'n_zeros_between' and at last another positive integer 'suffix'.
00205     The conventional element ID is xy(a)x'y'(a') with xya = the node ID in pier x, floor y and offgrid parameter a (optional);
00206         x'y'a' = the node ID in pier x', floor y' and offgrid parameter a' (optional).
00207     For more information on x and y, see GridIDConvention; for more information on a, see OffsetNodeIDConvention.
00208
00209     @param prefix (int): Prefix of the new ID. For a vertical element it should be the left node ID;
```

```
        for an horizontal one it should be the bottom node.
    @param suffix (int): Suffix of the new ID. For a vertical element it should be the right node ID;
        for an horizontal one it should be the top node.
    @param n_zeros_between (int, optional): Number of zeros to add between the two nodes. Defaults to
0.

    @exception NegativeValue: The argument prefix needs to be a positive integer.
    @exception NegativeValue: The argument suffix needs to be a positive integer.
    @exception NegativeValue: The argument n_zeros_between needs to be a positive integer.

    @returns int: The combined ID
    """
    # Convention:
    #       ElementID:      xy(a)x'y'(a')   with xy(a) = NodeID i and x'y'(a') = NodeID j
    #       TrussID:        xy(a)x'y'(a')   with xy(a) = NodeID i and x'y'(a') = NodeID j
    #       Spring:         xy(a)x'y'(a')   with xy(a) = NodeID i and x'y'(a') = NodeID j
    if prefix < 0: raise NegativeValue()
    if suffix < 0: raise NegativeValue()
    if n_zeros_between < 0: raise NegativeValue()

    return int(str(prefix*10**n_zeros_between) + str(suffix))


```

6.7.2.5 NodesOrientation()

```
def FunctionalFeatures.NodesOrientation (
            int iNode_ID,
            int jNode_ID )
```

Function that finds the orientation of the vector with direction 'jNode_ID''iNode_ID'.

If the the nodes are on top of each other, the function returns 'zero_length'.

Parameters

iNode_ID	(int): Node i.
jNode_ID	(int): Node j.

Exceptions

NegativeValue	The argument iNode_ID needs to be a positive integer.
NegativeValue	The argument jNode_ID needs to be a positive integer.

Returns

str: The orientation of the vector.

Definition at line 270 of file FunctionalFeatures.py.

```
def NodesOrientation(iNode_ID: int, jNode_ID: int):
    """
    Function that finds the orientation of the vector with direction 'jNode_ID''iNode_ID'.
    If the the nodes are on top of each other, the function returns 'zero_length'.

    @param iNode_ID (int): Node i.
    @param jNode_ID (int): Node j.

    @exception NegativeValue: The argument iNode_ID needs to be a positive integer.
    @exception NegativeValue: The argument jNode_ID needs to be a positive integer.

    @returns str: The orientation of the vector.
    """
```

```
    if iNode_ID < 1: raise NegativeValue()
    if jNode_ID < 1: raise NegativeValue()

    iNode = np.array(nodeCoord(iNode_ID))
    jNode = np.array(nodeCoord(jNode_ID))
    if abs(iNode[0]-jNode[0]) + abs(iNode[1]-jNode[1]) == 0:
        return "zero_length"
    elif abs(iNode[0]-jNode[0]) < abs(iNode[1]-jNode[1]):
        return "vertical"
    else:
        return "horizontal"


```

6.7.2.6 OffsetNodeIDConvention()

```
def FunctionalFeatures.OffsetNodeIDConvention (
            int node_ID,
            str orientation,
            str position_i_or_j )
```

Function used to add node on top of existing ones in the extremes of memebers with springs.

Parameters

node_ID	(int): Node that we refer to.
orientation	(str): Orientation of the memeber. Can be 'vertical' or 'horizontal'.
position_i_↩ or_j	(str): Position at the start 'i' (left or bottom) or at the end 'j' (right or top) of 'node_ID' in the member.

Exceptions

NegativeValue	The argument node_ID needs to be a positive integer.
WrongArgument	The argument position_i_or_j needs to be 'i' or 'j'
WrongArgument	The argument orientation needs to be 'vertical' or 'horizontal'

Returns

int: The combined ID

Definition at line 232 of file FunctionalFeatures.py.

```
00232 def OffsetNodeIDConvention(node_ID: int, orientation: str, position_i_or_j: str):
00233     """
00234     Function used to add node on top of existing ones in the extremes of memebers with springs.
00235 
00236     @param node_ID (int): Node that we refer to.
00237     @param orientation (str): Orientation of the memeber. Can be 'vertical' or 'horizontal'.
00238     @param position_i_or_j (str): Position at the start 'i' (left or bottom)
00239         or at the end 'j' (right or top) of 'node_ID' in the member.
00240 
00241     @exception NegativeValue: The argument node_ID needs to be a positive integer.
00242     @exception WrongArgument: The argument position_i_or_j needs to be 'i' or 'j'
00243     @exception WrongArgument: The argument orientation needs to be 'vertical' or 'horizontal'
00244 
00245     @returns int: The combined ID
00246     """
00247     # Convention:
      o xy
00248     #       GridNodeID:      xy              with x = pier, y = floor
      o xy7
```

```
    #   AdditionalNodeID:    xya           with x = pier, y = floor, a:  o xy  xy2 o-------o x'y3
x'y o    |
    #   PanelZoneNodeID:     xy(a)a      see MemberModel for the panel zone ID convention
      |
    #
      |
    #
      o xy'6
    #
      o xy'
    if node_ID < 1: raise NegativeValue()
    if position_i_or_j != "i" and position_i_or_j != "j": raise WrongArgument()

    if orientation == "vertical":
        if position_i_or_j == "i":
            return IDConvention(node_ID, 6)
        else:
            return IDConvention(node_ID, 7)
    elif orientation == "horizontal":
        if position_i_or_j == "i":
            return IDConvention(node_ID, 2)
        else:
            return IDConvention(node_ID, 3)
    else: raise WrongArgument()


```

6.7.2.7 plot_member()

```
def FunctionalFeatures.plot_member (
              list element_array,
               member_name = "Member name not defined",
               show_element_ID = True,
               show_node_ID = True )
```

Function that plots a set of elements.

It can be used to check the correctness of a part of the model or of a member. If the entire model need to be plotted, use instead 'plot_model("nodes", "elements")' from openseespy.postprocessing.Get_Rendering.
Inspired by plot_model written by Anurag Upadhyay and Christian Slotboom.

Parameters

element_array	(list): An array (list of lists of one dimensions and length = 3) that store the element and nodes IDs. An element is stored in one list with 3 entries: the element ID, node i ID and node j ID.
member_name	(str, optional): The name of what is plotted. Defaults to "Member name not defined".
show_element_ID	(bool, optional): Option to show the element IDs. Defaults to True.
show_node_ID	(bool, optional): Option to show the nodes IDs. Defaults to True.

Exceptions

WrongDimension	element_array needs to be non-empty.
WrongDimension	The number of entries in the lists inside the argument element_array need to be 3.

Returns

matplotlib.axes._subplots.AxesSubplo: The figure's wrappr, useful to customise the plot (change axes label, etc).

Definition at line 296 of file FunctionalFeatures.py.

```
00296 def plot_member(element_array: list, member_name = "Member name not defined", show_element_ID = True, show_node_ID = True):
00297     """
00298     Function that plots a set of elements. It can be used to check the correctness of a part of the model or of a member.
00299     If the entire model need to be plotted, use instead 'plot_model("nodes", "elements")' from openseespy.postprocessing.Get_Rendering. \n
00300     Inspired by plot_model written by Anurag Upadhyay and Christian Slotboom.
00301 
00302     @param element_array (list): An array (list of lists of one dimensions and length = 3) that store the element and nodes IDs.
00303         An element is stored in one list with 3 entries: the element ID, node i ID and node j ID.
00304     @param member_name (str, optional): The name of what is plotted. Defaults to "Member name not defined".
00305     @param show_element_ID (bool, optional): Option to show the element IDs. Defaults to True.
00306     @param show_node_ID (bool, optional): Option to show the nodes IDs. Defaults to True.
00307 
00308     @exception WrongDimension: element_array needs to be non-empty.
00309     @exception WrongDimension: The number of entries in the lists inside the argument element_array need to be 3.
00310 
00311     @returns matplotlib.axes._subplots.AxesSubplo: The figure's wrappr, useful to customise the plot (change axes label, etc).
00312     """
00313     if len(element_array) == 0: raise WrongArgument()
00314     if len(element_array[0]) != 3: raise WrongDimension()
00315 
00316     node_style = {'color':'black', 'marker':'o', 'facecolor':'black','linewidth':0.}
00317     node_text_style = {'fontsize':8, 'fontweight':'regular', 'color':'green'}
00318     track_node = {}
00319 
00320     if show_element_ID:
00321         show_e_ID = 'yes'
00322     else:
00323         show_e_ID = 'no'
00324 
00325     fig = plt.figure()
00326     ax = fig.add_subplot(1,1,1)
00327 
00328     for ele in element_array:
00329         eleTag = int(ele[0])
00330         Nodes =ele[1:]
00331 
00332         if len(Nodes) == 2:
00333             # 2D element
00334             iNode = np.array(nodeCoord(Nodes[0]))
00335             jNode = np.array(nodeCoord(Nodes[1]))
00336             ipltf._plotBeam2D(iNode, jNode, ax, show_e_ID, eleTag, "solid")
00337             ax.scatter(*iNode, **node_style)
00338             ax.scatter(*jNode, **node_style)
00339             if show_node_ID:
00340                 if abs(sum(iNode - jNode)) > 1e-6:
00341                     # beam-col
00342                     __plt_node(Nodes[0], track_node, iNode, ax, node_text_style)
00343                     __plt_node(Nodes[1], track_node, jNode, ax, node_text_style, h_align='right', v_align='bottom')
00344                 else:
00345                     # zerolength
00346                     __plt_node(Nodes[0], track_node, iNode, ax, node_text_style, h_align='right')
00347                     __plt_node(Nodes[1], track_node, jNode, ax, node_text_style)
00348         else:
00349             print("Too many nodes in this elemnet (see shell elements)")
00350 
00351     ax.set_xlabel('x [{}]'.format(length_unit))
00352     ax.set_ylabel('y [{}]'.format(length_unit))
00353     plt.title("Visualisation of: {}".format(member_name))
00354     plt.axis('equal')
00355     return ax
00356 
00357 
```

6.7.2.8 plot_nodes()

```
def FunctionalFeatures.plot_nodes (
            list nodes_array,
             name = "Not defined",
             show_node_ID = True )
```

Function that plots a set of nodes.

It can be used to check the correctness of the model's geometry. If the entire model need to be plotted, use instead 'plot_model("nodes", "elements")' from openseespy.postprocessing.Get_Rendering.

Parameters

nodes_array	(list): List of 1 dimension with the IDs of the nodes to be displayed.
name	(str, optional): Name that describe what the plot will show. Defaults to "Not defined".
show_node_ID	(bool, optional): Option to show the node IDs. Defaults to True.

Exceptions

WrongArgument	nodes_array needs to be non-empty.

Returns

(matplotlib.axes._subplots.AxesSubplot): The figure's wrapper, useful to customise the plot (change axes label, etc).

Definition at line 358 of file FunctionalFeatures.py.

```
00358 def plot_nodes(nodes_array: list, name = "Not defined", show_node_ID = True):
00359     """
00360     Function that plots a set of nodes. It can be used to check the correctness of the model's
      geometry.
00361     If the entire model need to be plotted, use instead 'plot_model("nodes", "elements")' from
      openseespy.postprocessing.Get_Rendering.
00362 
00363     @param nodes_array (list): List of 1 dimension with the IDs of the nodes to be displayed.
00364     @param name (str, optional): Name that describe what the plot will show. Defaults to "Not
      defined".
00365     @param show_node_ID (bool, optional): Option to show the node IDs. Defaults to True.
00366 
00367     @exception WrongArgument: nodes_array needs to be non-empty.
00368 
00369     @returns (matplotlib.axes._subplots.AxesSubplot): The figure's wrapper, useful to customise the
      plot (change axes label, etc).
00370     """
00371     if len(nodes_array) == 0: raise WrongArgument()
00372 
00373     node_style = {'color':'black', 'marker':'o', 'facecolor':'black','linewidth':0.}
00374     node_text_style = {'fontsize':8, 'fontweight':'regular', 'color':'green'}
00375     track_node = {}
00376 
00377     fig = plt.figure()
00378     ax = fig.add_subplot(1,1,1)
00379 
00380     for node_ID in nodes_array:
00381         node_xy = np.array(nodeCoord(node_ID))
00382         ax.scatter(*node_xy, **node_style)
00383         if show_node_ID:
00384             __plt_node(node_ID, track_node, node_xy, ax, node_text_style)
00385 
00386     ax.set_xlabel('x [{}]'.format(length_unit))
00387     ax.set_ylabel('y [{}]'.format(length_unit))
00388     plt.title("Visualisation of: {}".format(name))
00389     plt.axis('equal')
00390     return ax
00391 
00392 
```

6.7.2.9 ProgressingPercentage()

```
def FunctionalFeatures.ProgressingPercentage (
            max_iter,
```

```
int i,
int next_step,
 step = 10 )
```

Function that shows the progressing percentage of an iterative process.

Parameters

max_iter	(int): Maximal number of interations
i	(int): Current iteration
next_step	(int): Next step of the percentage (set to 0 for the first iteration and then use the return parameter)
step	(int, optional): Size of the step (should be a fraction of 100). Defaults to 10.

Returns

int: The updated next step

Definition at line 14 of file FunctionalFeatures.py.

```
00014 def ProgressingPercentage(max_iter, i: int, next_step: int, step = 10):
00015     """
00016     Function that shows the progressing percentage of an iterative process.
00017
00018     @param max_iter (int): Maximal number of interations
00019     @param i (int): Current iteration
00020     @param next_step (int): Next step of the percentage (set to 0 for the first iteration and then use
      the return parameter)
00021     @param step (int, optional): Size of the step (should be a fraction of 100). Defaults to 10.
00022
00023     @returns int: The updated next step
00024     """
00025     if i*100.0/(max_iter-1) >= next_step:
00026         print("The progression is {}%".format(next_step))
00027         return next_step + step
00028
00029     return next_step
00030
00031
```

6.8 GeometryTemplate Namespace Reference

Module with geometry templates (nodes and/or elements with associated fibers, material models, etc).

Functions

- def DefineFrameNodes (int n_hor_axis, int n_vert_axis, storey_width, storey_height, half_pz_height=np.↩ array([ ]), origin=[0, 0], first_hor_axis=1, first_vert_axis=1, show_plot=True)

  Function that declares and initialises the grid nodes of a frame.
- def DefineFrameNodesAndElementsSteelIShape (int n_hor_axis, int n_vert_axis, storey_width, storey_↩ height, list list_col, list list_beam, int geo_trans_ID, N_G=np.array([ ]), t_dp=np.array([ ]), L_b_col=np.array([ ]), L_b_beam=np.array([ ]), fix_support=True, show_plot=True, panel_zone=True)

  WIP (Work In Progress).
- def DefineSubassemblageNodes (beam_left_L_cl, beam_right_L_cl, col_top_L_cl, col_bottom_L_cl, depth↩ _col, depth_beam, boundary_condition=True, show_plot=True)

  Function that declares and initialises the grid nodes of an interior subassemblage.
- def Initialize2DModel (data_dir="Results")

  Function that initialise the project creating the 2D model with 3 DOF per node and set up a directory for the results.

6.8.1 Detailed Description

Module with geometry templates (nodes and/or elements with associated fibers, material models, etc).

Carmine Schipani, 2021

6.8.2 Function Documentation

6.8.2.1 DefineFrameNodes()

```
def GeometryTemplate.DefineFrameNodes (
             int n_hor_axis,
             int n_vert_axis,
              storey_width,
              storey_height,
              half_pz_height = np.array([]),
              origin = [0, 0],
              first_hor_axis = 1,
              first_vert_axis = 1,
              show_plot = True )
```

Function that declares and initialises the grid nodes of a frame.

Option to offset the grid node of the panel zones with the master node of the panel zone being the grid one (top center one). The function can be used multiple times to create more complex geometries.

Parameters

n_hor_axis	(int): Number of horizontal axis (or piers) for the grid of the frame.
n_vert_axis	(int): Number of vertical axis (or floors) for the grid of the frame.
storey_width	(float): Width of the bays.
storey_height	(float): Height of the storeys.
half_pz_height	(np.ndarray, optional): Array of 1 dimension with half the height of the panel zone for each floor. The first floor should be 0 (no panel zone in the support). Defaults to np.array([ ]), e.g. no panel zone.
origin	(list, optional): List of two entry with the origin position. Defaults to [0, 0].
first_hor_axis	(int, optional): Number of the first pier. Defaults to 1.
first_vert_axis	(int, optional): Number of the first floor. Defaults to 1.
show_plot	(bool, optional): Option to show the plot of the nodes declared and initialised. Defaults to True.

Exceptions

NegativeValue	n_hor_axis needs to be a positive integer.
NegativeValue	n_vert_axis needs to be a positive integer.
NegativeValue	storey_width needs to be positive.
NegativeValue	storey_height needs to be positive.
WrongDimension	origin has a dimension of 2.
NegativeValue	first_hor_axis needs to be a positive integer.

Exceptions

NegativeValue	first_vert_axis needs to be a positive integer.
WrongDimension	size of half_pz_height needs to be equal to n_vert_axis, if different from 0.

Returns

list: List with the nodes declared.

Definition at line 44 of file GeometryTemplate.py.

```
00045     origin = [0, 0], first_hor_axis = 1, first_vert_axis = 1, show_plot = True):
00046     """
00047     Function that declares and initialises the grid nodes of a frame. Option to offset the grid node
   of the panel zones
00048         with the master node of the panel zone being the grid one (top center one). The function can
   be used multiple times
00049         to create more complex geometries.
00050
00051     @param n_hor_axis (int): Number of horizontal axis (or piers) for the grid of the frame.
00052     @param n_vert_axis (int): Number of vertical axis (or floors) for the grid of the frame.
00053     @param storey_width (float): Width of the bays.
00054     @param storey_height (float): Height of the storeys.
00055     @param half_pz_height (np.ndarray, optional): Array of 1 dimension with half the height of the
   panel zone for each floor.
00056         The first floor should be 0 (no panel zone in the support). Defaults to np.array([]), e.g. no
   panel zone.
00057     @param origin (list, optional): List of two entry with the origin position. Defaults to [0, 0].
00058     @param first_hor_axis (int, optional): Number of the first pier. Defaults to 1.
00059     @param first_vert_axis (int, optional): Number of the first floor. Defaults to 1.
00060     @param show_plot (bool, optional): Option to show the plot of the nodes declared and initialised.
   Defaults to True.
00061
00062     @exception NegativeValue: n_hor_axis needs to be a positive integer.
00063     @exception NegativeValue: n_vert_axis needs to be a positive integer.
00064     @exception NegativeValue: storey_width needs to be positive.
00065     @exception NegativeValue: storey_height needs to be positive.
00066     @exception WrongDimension: origin has a dimension of 2.
00067     @exception NegativeValue: first_hor_axis needs to be a positive integer.
00068     @exception NegativeValue: first_vert_axis needs to be a positive integer.
00069     @exception WrongDimension: size of half_pz_height needs to be equal to n_vert_axis, if different
   from 0.
00070
00071     @returns list: List with the nodes declared.
00072     """
00073     if n_hor_axis < 1: raise NegativeValue()
00074     if n_vert_axis < 1: raise NegativeValue()
00075     if storey_width < 0: raise NegativeValue()
00076     if storey_height < 0: raise NegativeValue()
00077     if len(origin) != 2: raise WrongDimension()
00078     if first_hor_axis < 1: raise NegativeValue()
00079     if first_vert_axis < 1: raise NegativeValue()
00080     if np.size(half_pz_height) != 0 and np.size(half_pz_height) != n_vert_axis: raise WrongDimension()
00081
00082     if np.size(half_pz_height) == 0: half_pz_height = np.zeros(n_vert_axis)
00083     node_array = []
00084     max_n_x = n_hor_axis + first_hor_axis - 1
00085     max_n_y = n_vert_axis + first_vert_axis - 1
00086     for xx in range(n_hor_axis):
00087         x_axis = xx + first_hor_axis
00088         for yy in range(n_vert_axis):
00089             y_axis = yy + first_vert_axis
00090             node_ID = GridIDConvention(x_axis, y_axis, max_n_x, max_n_y)
00091             node(node_ID, origin[0]+xx*storey_width, origin[1]+yy*storey_height+half_pz_height[yy])
00092             if y_axis == 1 and half_pz_height[yy] != 0: print("Warning: the nodes at the base have a
   panel zone height")
00093             node_array.append(node_ID)
00094
00095     if show_plot:
00096         plot_nodes(node_array, "Frame geometry template with only nodes", True)
00097         plt.grid()
00098
00099     return node_array
00100
00101
```

6.8.2.2 DefineFrameNodesAndElementsSteelIShape()

```
def GeometryTemplate.DefineFrameNodesAndElementsSteelIShape (
            int n_hor_axis,
            int n_vert_axis,
             storey_width,
             storey_height,
            list list_col,
            list list_beam,
            int geo_trans_ID,
             N_G = np.array([]),
             t_dp = np.array([]),
             L_b_col = np.array([]),
             L_b_beam = np.array([]),
             fix_support = True,
             show_plot = True,
             panel_zone = True )
```

WIP (Work In Progress).

Function that declares and initialises the grid nodes of a frame and the members using steel I shape SpringBasedElements. WARNING: Current limit of the geometry: n_hor_axis and n_vert_axis $<$ 10; if exceeded, there are problems with the IDs (ID limit is exceeded, $\sim$2.2e9). WARNING: if the section of the columns change, the function does not account for the splacing. Each colum section is defined from floor to floor; if there is a change in the column section, it happens right after the panel zone (not realistic but good enough for predesign). WIP: Solve ID limit for large building need implementations (for example the use of a different ID convention or the use of the class IDGenerator).

Parameters

n_hor_axis	(int): Number of horizontal axis (or piers) for the grid of the frame.
n_vert_axis	(int): Number of vertical axis (or floors) for the grid of the frame.
storey_width	(float): Width of the bays.
storey_height	(float): Height of the storeys.
list_col	(list(SteelIShape)): List with the sections of the columns for every floor.
list_beam	(list(SteelIShape)): List with the sections of the beams for every bay.
geo_trans_ID	(int): The geometric transformation (for more information, see OpenSeesPy documentation).
N_G	(np.ndarray, optional): Array of dimension 1 with the axial load for each column (starting at floor 2). Defaults to np.array([ ]), e.g. 0.
t_dp	(np.ndarray, optional): Array of dimension 1 with the doubler plate thickness for each bay's beam. Defaults to np.array([ ]), e.g. 0.
L_b_col	(np.ndarray, optional): Array of dimension 1 with the maxiaml unbraced lateral buckling length for each column. Defaults to np.array([ ]), e.g. -1.
L_b_beam	(np.ndarray, optional): Array of dimension 1 with the maxiaml unbraced lateral buckling length for each beam. Defaults to np.array([ ]), e.g. -1.
fix_support	(bool, optional): Option to fix the support of the frame. Defaults to True.
show_plot	(bool, optional): Option to show the plot of the nodes declared and initialised. Defaults to True.
panel_zone	(bool, optional): Option to add the panel zones in the model. Defaults to True.

Exceptions

WrongDimension	N_G dimension needs to be equal to n_vert_axis-1, if different from 0.
WrongDimension	t_dp dimension needs to be equal to n_vert_axis-1, if different from 0.

Exceptions

WrongDimension	L_b_col dimension needs to be equal to n_vert_axis-1, if different from 0.
WrongDimension	L_b_beam dimension needs to be equal to n_hor_axis-1, if different from 0.
WrongDimension	list_col dimension needs to be equal to n_vert_axis-1.
WrongDimension	list_beam dimension needs to be equal to n_vert_axis-1.
NegativeValue	geo_trans_ID needs to be a positive integer.

Returns

List: List with the element objects in the frame.

Definition at line 102 of file GeometryTemplate.py.

```
    fix_support = True, show_plot = True, panel_zone = True):
    """
    WIP (Work In Progress). Function that declares and initialises the grid nodes of a frame and the members using steel I shape SpringBasedElements.
    WARNING: Current limit of the geometry: n_hor_axis and n_vert_axis < 10; if exceeded, there are problems with the IDs (ID limit is exceeded, ~2.2e9).
    WARNING: if the section of the columns change, the function does not account for the splacing. Each colum section is defined from floor to floor;
      if there is a change in the column section, it happens right after the panel zone (not realistic but good enough for predesign).
    WIP: Solve ID limit for large building need implementations (for example the use of a different ID convention or the use of the class IDGenerator).

    @param n_hor_axis (int): Number of horizontal axis (or piers) for the grid of the frame.
    @param n_vert_axis (int): Number of vertical axis (or floors) for the grid of the frame.
    @param storey_width (float): Width of the bays.
    @param storey_height (float): Height of the storeys.
    @param list_col (list(SteelIShape)): List with the sections of the columns for every floor.
    @param list_beam (list(SteelIShape)): List with the sections of the beams for every bay.
    @param geo_trans_ID (int): The geometric transformation (for more information, see OpenSeesPy documentation).
    @param N_G (np.ndarray, optional): Array of dimension 1 with the axial load for each column (starting at floor 2). Defaults to np.array([]), e.g. 0.
    @param t_dp (np.ndarray, optional): Array of dimension 1 with the doubler plate thickness for each bay's beam. Defaults to np.array([]), e.g. 0.
    @param L_b_col (np.ndarray, optional): Array of dimension 1 with the maxiaml unbraced lateral buckling length for each column. Defaults to np.array([]), e.g. -1.
    @param L_b_beam (np.ndarray, optional): Array of dimension 1 with the maxiaml unbraced lateral buckling length for each beam. Defaults to np.array([]), e.g. -1.
    @param fix_support (bool, optional): Option to fix the support of the frame. Defaults to True.
    @param show_plot (bool, optional): Option to show the plot of the nodes declared and initialised. Defaults to True.
    @param panel_zone (bool, optional): Option to add the panel zones in the model. Defaults to True.

    @exception WrongDimension: N_G dimension needs to be equal to n_vert_axis-1, if different from 0.
    @exception WrongDimension: t_dp dimension needs to be equal to n_vert_axis-1, if different from 0.
    @exception WrongDimension: L_b_col dimension needs to be equal to n_vert_axis-1, if different from 0.
    @exception WrongDimension: L_b_beam dimension needs to be equal to n_hor_axis-1, if different from 0.
    @exception WrongDimension: list_col dimension needs to be equal to n_vert_axis-1.
    @exception WrongDimension: list_beam dimension needs to be equal to n_vert_axis-1.
    @exception NegativeValue: geo_trans_ID needs to be a positive integer.

    @returns List: List with the element objects in the frame.
    """
    panel_zone = True
    if np.size(N_G) == 0: N_G = np.zeros(n_vert_axis-1)
    if np.size(t_dp) == 0: t_dp = np.zeros(n_vert_axis-1)
    if np.size(L_b_col) == 0: L_b_col = np.ones(n_vert_axis-1) * (-1.0)
    if np.size(L_b_beam) == 0: L_b_beam = np.ones(n_hor_axis-1) * (-1.0)

    if np.size(list_col) != n_vert_axis-1: raise WrongDimension()
    if np.size(list_beam) != n_vert_axis-1: raise WrongDimension()
    if np.size(N_G) != n_vert_axis-1: raise WrongDimension()
    if np.size(t_dp) != n_vert_axis-1: raise WrongDimension()
    if np.size(L_b_col) != n_vert_axis-1: raise WrongDimension()
    if np.size(L_b_beam) != n_hor_axis-1: raise WrongDimension()
    if geo_trans_ID < 1: raise NegativeValue()

    half_pz_height = np.zeros(n_vert_axis)
    if panel_zone:
        for ii, beam in enumerate(list_beam):
            half_pz_height[ii+1] = beam.d/2
```

```

    node_array = DefineFrameNodes(n_hor_axis, n_vert_axis, storey_width, storey_height, half_pz_height, [0, 0], 1, 1, False)

    beam_column_pzspring = [[], [], []]
    for xx in range(n_hor_axis):
        for yy in range(n_vert_axis):
            node_ID = node_array[xx*n_vert_axis + yy]
            if yy != 0:
                # Panel Zone
                if half_pz_height[yy] == 0:
                    col_j_node_ID = node_ID
                    beam_j_node_ID = node_ID
                else:
                    tmp_pz = PanelZoneSteelIShapeSkiadopoulos2021(node_ID, list_col[yy-1], list_beam[yy-1], geo_trans_ID, t_dp[yy-1])
                    tmp_pz.CreateMember()
                    col_j_node_ID = IDConvention(node_ID, 5, 1)
                    beam_j_node_ID = IDConvention(node_ID, 8, 1)
                    beam_column_pzspring[2].append(deepcopy(tmp_pz))

                # Column
                col_i_node_ID = node_array[xx*n_vert_axis + yy - 1]
                col_mat_i = OffsetNodeIDConvention(col_i_node_ID, "vertical", "i")
                col_mat_j = OffsetNodeIDConvention(col_j_node_ID, "vertical", "j")
                ele_ID = col_i_node_ID if panel_zone else -1
                tmp_col = SpringBasedElementModifiedIMKSteelIShape(col_i_node_ID, col_j_node_ID, list_col[yy-1], geo_trans_ID,
                    col_mat_i, col_mat_j, N_G[yy-1], L_b=L_b_col[yy-1], ele_ID = ele_ID)
                tmp_col.CreateMember()
                beam_column_pzspring[1].append(deepcopy(tmp_col))

                if xx != 0:
                    # Beam
                    if half_pz_height[yy] == 0:
                        beam_i_node_ID = node_array[(xx-1)*n_vert_axis + yy]
                    else:
                        beam_i_node_ID = IDConvention(node_array[(xx-1)*n_vert_axis + yy], 2, 1)
                    beam_mat_i = OffsetNodeIDConvention(beam_i_node_ID, "horizontal", "i")
                    beam_mat_j = OffsetNodeIDConvention(beam_j_node_ID, "horizontal", "j")
                    ele_ID = beam_i_node_ID if panel_zone else -1
                    tmp_beam = SpringBasedElementModifiedIMKSteelIShape(beam_i_node_ID, beam_j_node_ID, list_beam[yy-1], geo_trans_ID,
                        beam_mat_i, beam_mat_j, L_b=L_b_beam[xx-1], ele_ID = ele_ID)
                    tmp_beam.CreateMember()
                    beam_column_pzspring[0].append(deepcopy(tmp_beam))
            else:
                if fix_support: RigidSupport(node_ID)


    if show_plot:
        opsplt.plot_model("nodes", "elements")

    return beam_column_pzspring


```

6.8.2.3 DefineSubassemblageNodes()

def GeometryTemplate.DefineSubassemblageNodes (
beam_left_L_cl,
beam_right_L_cl,
col_top_L_cl,
col_bottom_L_cl,
depth_col,
depth_beam,
boundary_condition = True,
show_plot = True)

Function that declares and initialises the grid nodes of an interior subassemblage.

The panel zone geometry is defined by the two arguments depth_col and depth_beam.

Parameters

beam_left_L_cl	(float): Centerline length of the left beam (excluding the panel zone).
beam_right_L_cl	(float): Centerline length of the right beam (excluding the panle zone).
col_top_L_cl	(float): Centerline length of the top column (excluding the panel zone).
col_bottom_L_cl	(float): Centerline length of the bottom column (excluding the panel zone).
depth_col	(float): Depth of the columns for the panel zone.
depth_beam	(float): Depth of the beams for the panel zone.
boundary_condition	(bool, optional): Option to set already the boundary condition (bottom column pinned, beams fix only y movement). Defaults to True.
show_plot	(bool, optional): Option to show the plot of the nodes declared and initialised. Defaults to True.

Exceptions

NegativeValue	beam_left_L_cl needs to be positive.
NegativeValue	beam_right_L_cl needs to be positive.
NegativeValue	col_top_L_cl needs to be positive.
NegativeValue	col_bottom_L_cl needs to be positive.
NegativeValue	depth_col needs to be positive.
NegativeValue	depth_beam needs to be positive.

Returns

list: List with the nodes declared.

Definition at line 207 of file GeometryTemplate.py.

```
    boundary_condition = True, show_plot = True):
    """
    Function that declares and initialises the grid nodes of an interior subassemblage. The panel zone geometry is defined by the two arguments
        depth_col and depth_beam.

    @param beam_left_L_cl (float): Centerline length of the left beam (excluding the panel zone).
    @param beam_right_L_cl (float): Centerline length of the right beam (excluding the panle zone).
    @param col_top_L_cl (float): Centerline length of the top column (excluding the panel zone).
    @param col_bottom_L_cl (float): Centerline length of the bottom column (excluding the panel zone).
    @param depth_col (float): Depth of the columns for the panel zone.
    @param depth_beam (float): Depth of the beams for the panel zone.
    @param boundary_condition (bool, optional): Option to set already the boundary condition (bottom column pinned, beams fix only y movement).
        Defaults to True.
    @param show_plot (bool, optional): Option to show the plot of the nodes declared and initialised. Defaults to True.

    @exception NegativeValue: beam_left_L_cl needs to be positive.
    @exception NegativeValue: beam_right_L_cl needs to be positive.
    @exception NegativeValue: col_top_L_cl needs to be positive.
    @exception NegativeValue: col_bottom_L_cl needs to be positive.
    @exception NegativeValue: depth_col needs to be positive.
    @exception NegativeValue: depth_beam needs to be positive.

    @returns list: List with the nodes declared.
    """
    # origin is the bottom left corner
    if beam_left_L_cl < 0: raise NegativeValue()
    if beam_right_L_cl < 0: raise NegativeValue()
    if col_top_L_cl < 0: raise NegativeValue()
    if col_bottom_L_cl < 0: raise NegativeValue()
    if depth_col < 0: raise NegativeValue()
    if depth_beam < 0: raise NegativeValue()

    node(12, 0.0, col_bottom_L_cl+depth_beam/2)
    node(21, beam_left_L_cl+depth_col/2, 0.0)
    node(22, beam_left_L_cl+depth_col/2, col_bottom_L_cl+depth_beam)
    node(23, beam_left_L_cl+depth_col/2, col_bottom_L_cl+depth_beam+col_top_L_cl)
```

```
    node(32, beam_left_L_cl+depth_col+beam_right_L_cl, col_bottom_L_cl+depth_beam/2)
    node_array = [12, 21, 22, 23, 32]

    if boundary_condition:
        fix(12, 0, 1, 0)
        fix(32, 0, 1, 0)
        fix(21, 1, 1, 0)

    if show_plot:
        plot_nodes(node_array, "Subassemblage geometry template with only nodes", True)
        plt.grid()

    return node_array


# def DefineRCSSubassemblage():
#     # WIP and experimental


```

6.8.2.4 Initialize2DModel()

```
def GeometryTemplate.Initialize2DModel (
                data_dir = "Results" )
```

Function that initialise the project creating the 2D model with 3 DOF per node and set up a directory for the results.

Parameters

data_dir	(str, optional): Directory where the data will be stored. The function forces the user to define it just for good practice and consistency between projects. Defaults to "Results".

Definition at line 25 of file GeometryTemplate.py.

```
00025 def Initialize2DModel(data_dir = "Results"):
00026     """
00027     Function that initialise the project creating the 2D model with 3 DOF per node and set up a
       directory for the results.
00028
00029     @param data_dir (str, optional): Directory where the data will be stored.
00030         The function forces the user to define it just for good practice and consistency between
       projects.
00031         Defaults to "Results".
00032     """
00033     # Clear all
00034     wipe()
00035
00036     # Build model (2D - 3 DOF/node)
00037     model('basic', '-ndm', 2, '-ndf', 3)
00038
00039     # Main Results Folder
00040     if not os.path.exists(data_dir):
00041         os.makedirs(data_dir)
00042
00043
```

6.9 MaterialModels Namespace Reference

Module for the material models.

Classes

- class ConfMander1988Circ

Class that stores funcions and material properties of a RC circular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

- class ConfMander1988CircRCCircShape

  Class that is the children of ConfMander1988Circ and combine the class RCCircShape (section) to retrieve the information needed.
- class ConfMander1988Rect

  Class that stores funcions and material properties of a RC rectangular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.
- class ConfMander1988RectRCRectShape

  Class that is the children of ConfMander1988Rect and combine the class RCRectShape (section) to retrieve the information needed.
- class GMP1970

  Class that stores funcions and material properties of the vertical steel reinforcement bars with Giuffré, Menegotto and Pinto 1970 as the material model and the OpenSeesPy command type used to model it is Steel02.
- class GMP1970RCRectShape

  Class that is the children of GMP1970 and combine the class RCRectShape (section) to retrieve the information needed.
- class Gupta1999

  Class that stores funcions and material properties of a steel double symmetric I-shape profile with Gupta 1999 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.
- class Gupta1999SteelIShape

  Class that is the children of Gupta1999 and combine the class SteelIShape (section) to retrieve the information needed.
- class MaterialModels

  Parent abstract class for the storage and manipulation of a material model's information (mechanical and geometrical parameters, etc) and initialisation in the model.
- class ModifiedIMK

  Class that stores funcions and material properties of a steel double symmetric I-shape profile with modified Ibarra-↵ Medina-Krawinkler as the material model for the nonlinear springs and the OpenSeesPy command type used to model it is Bilin.
- class ModifiedIMKSteelIShape

  Class that is the children of ModifiedIMK and combine the class SteelIShape (section) to retrieve the information needed.
- class Skiadopoulos2021

  Class that stores funcions and material properties of a steel double symmetric I-shape profile with Skiadopoulos 2021 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.
- class Skiadopoulos2021RCS

  WIP: Class that is the children of Skiadopoulos2021 and it's used for the panel zone spring in a RCS (RC column continous, Steel beam).
- class Skiadopoulos2021SteelIShape

  Class that is the children of Skiadopoulos2021 and combine the class SteelIShape (section) to retrieve the information needed.
- class UnconfMander1988

  Class that stores funcions and material properties of a RC rectangular or circular section with Mander 1988 as the material model for the unconfined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.
- class UnconfMander1988RCCircShape

  Class that is the children of UnconfMander1988 and combine the class RCCircShape (section) to retrieve the information needed.
- class UnconfMander1988RCRectShape

  Class that is the children of UnconfMander1988 and combine the class RCRectShape (section) to retrieve the information needed.
- class UniaxialBilinear

Class that stores funcions and material properties of a simple uniaxial bilinear model with the OpenSeesPy command type used to model it is Steel01.

- class UniaxialBilinearSteelIShape

  Class that is the children of UniaxialBilinear and combine the class SteelIShape (section) to retrieve the information needed.

- class UVC

  Class that stores funcions and material properties of a steel profile or reinforcing bar with Updated Voce-Chaboche as the material model and the OpenSeesPy command type used to model it is UVCuniaxial.

- class UVCCalibrated

  Class that is the children of UVC that retrieve calibrated parameters from UVC_calibrated_parameters.txt.

- class UVCCalibratedRCCircShape

  Class that is the children of UVCCalibrated and combine the class RCCircShape (section) to retrieve the information needed.

- class UVCCalibratedRCRectShape

  Class that is the children of UVCCalibrated and combines the class RCRectShape (section) to retrieve the information needed.

- class UVCCalibratedSteelIShapeFlange

  Class that is the children of UVCCalibrated and combine the class SteelIShape (section) to retrieve the information needed for the material model of the flange (often used fo the entire section).

- class UVCCalibratedSteelIShapeWeb

  Class that is the children of UVCCalibrated and combine the class SteelIShape (section) to retrieve the information needed for the material model of the web.

Functions

- def Concrete01Funct (fc, ec, fpcu, ecu, discretized_eps)

  Function with the equation of the curve of the Concrete01 model.

- def Concrete04Funct (fc, discretized_eps, ec, Ec)

  Function with the equation of the curve of the confined and unconfined concrete (Popovics 1973).

- def PlotConcrete01 (fc, ec, fpcu, ecu, ax, ID=0)

  Function that plots the Concrete01 stress-strain curve.

- def PlotConcrete04 (fc, Ec, ec, ecu, str Type, ax, ID=0)

  Function that plots the confined/unconfined Concrete04 stress-strain curve.

6.9.1 Detailed Description

Module for the material models.

Carmine Schipani, 2021

6.9.2 Function Documentation

6.9.2.1 Concrete01Funct()

```
def MaterialModels.Concrete01Funct (
              fc,
              ec,
              fpcu,
              ecu,
              discretized_eps )
```

Function with the equation of the curve of the Concrete01 model.

For more information, see Kent-Scott-Park concrete material object with degraded linear unloading/reloading stiffness according to the work of Karsan-Jirsa and no tensile strength.

Parameters

fc	(float): Compressive concrete yield stress (negative).
ec	(float): Compressive concrete yield strain (negative).
fpcu	(float): Concrete crushing strength (negative).
ecu	(float): Concrete strain at crushing strength (negative).
discretized_eps	(float): Variable strain.

Returns

float: Stress in function of variable strain.

Definition at line 2969 of file MaterialModels.py.

```
def Concrete01Funct(fc, ec, fpcu, ecu, discretized_eps):
    """
    Function with the equation of the curve of the Concrete01 model.
    For more information, see Kent-Scott-Park concrete material object with
        degraded linear unloading/reloading stiffness according to the work of Karsan-Jirsa and no
 tensile strength.

    @param fc (float): Compressive concrete yield stress (negative).
    @param ec (float): Compressive concrete yield strain (negative).
    @param fpcu (float): Concrete crushing strength (negative).
    @param ecu (float): Concrete strain at crushing strength (negative).
    @param discretized_eps (float): Variable strain.

    @returns float: Stress in function of variable strain.
    """
    if discretized_eps > ec:
        eta = discretized_eps/ec;
        return fc*(2*eta-eta*eta);
    else:
        Ttangent = (fc-fpcu)/(ec-ecu)
        return fc + Ttangent*(discretized_eps-ec);


```

6.9.2.2 Concrete04Funct()

```
def MaterialModels.Concrete04Funct (
                fc,
                discretized_eps,
                ec,
                Ec )
```

Function with the equation of the curve of the confined and unconfined concrete (Popovics 1973).

Parameters

fc	(float): Compressive concrete yield stress (negative).
discretized_eps	(float): Variable strain.
ec	(float): Compressive concrete yield strain (negative).
Ec	(float): Concrete Young modulus.

Returns

float: Stress in function of variable strain.

Definition at line 2913 of file MaterialModels.py.

```
def Concrete04Funct(fc, discretized_eps, ec, Ec):
    """
    Function with the equation of the curve of the confined and unconfined concrete (Popovics 1973).

    @param fc (float): Compressive concrete yield stress (negative).
    @param discretized_eps (float): Variable strain.
    @param ec (float): Compressive concrete yield strain (negative).
    @param Ec (float): Concrete Young modulus.

    @returns float: Stress in function of variable strain.
    """
    x = discretized_eps/ec
    r = Ec / (Ec - fc/ec)
    return fc*x*r / (r-1+x**r)


```

6.9.2.3 PlotConcrete01()

```
def MaterialModels.PlotConcrete01 (
        fc,
        ec,
        fpcu,
        ecu,
        ax,
        ID = 0 )
```

Function that plots the Concrete01 stress-strain curve.

Parameters

fc	(float): Compressive concrete yield stress (negative).
ec	(float): Compressive concrete yield strain (negative).
fpcu	(float): Concrete crushing strength (negative).
ecu	(float): Concrete strain at crushing strength (negative).
ax	(matplotlib.axes._subplots.AxesSubplot): The figure's wrapper.
ID	(int, optional): ID of the material model. Defaults to 0 (= not defined).

Example: to create the plot, call this line to pass the correct ax: fig, ax = plt.subplots()

Definition at line 2991 of file MaterialModels.py.

```
def PlotConcrete01(fc, ec, fpcu, ecu, ax, ID = 0):
    """
    Function that plots the Concrete01 stress-strain curve.

    @param fc (float): Compressive concrete yield stress (negative).
    @param ec (float): Compressive concrete yield strain (negative).
    @param fpcu (float): Concrete crushing strength (negative).
    @param ecu (float): Concrete strain at crushing strength (negative).
    @param ax (matplotlib.axes._subplots.AxesSubplot): The figure's wrapper.
    @param ID (int, optional): ID of the material model. Defaults to 0 (= not defined).

    Example: to create the plot, call this line to pass the correct ax:
        fig, ax = plt.subplots()
    """

    # Data for plotting
    N = 1000
    x_axis = np.zeros(N)
    y_axis = np.zeros(N)
    for i in range(N):
        x_axis[i] = i/N*ecu
        y_axis[i] = Concrete01Funct(fc, ec, fpcu, ecu, x_axis[i])

```

```
    ax.plot(x_axis*100.0, y_axis/MPa_unit, 'k--', label = "Co01")
    ax.set(xlabel='Strain [%]', ylabel='Stress [MPa]',
                title='Mander 1988 (Concrete01) material model (ID={})'.format(ID))
    plt.legend()
    plt.grid()


# Private functions
```

6.9.2.4 PlotConcrete04()

```
def MaterialModels.PlotConcrete04 (
                fc,
                Ec,
                ec,
                ecu,
               str Type,
                ax,
                ID = 0 )
```

Function that plots the confined/unconfined Concrete04 stress-strain curve.

Parameters

fc	(float): Compressive concrete yield strength (needs to be negative).
Ec	(float): Young modulus.
ec	(float): Compressive concrete yield strain.
ecu	(float): Compressive concrete failure strain (negative).
Type	(str): Type of concrete (confined = 'C', unconfined = 'U')
ax	(matplotlib.axes._subplots.AxesSubplot): The figure's wrapper.
ID	(int, optional): ID of the material model. Defaults to 0 (= not defined).

Exceptions

NameError	

Example: to create the plot, call this line to pass the correct ax: fig, ax = plt.subplots()

Definition at line 2929 of file MaterialModels.py.

```
02929 def PlotConcrete04(fc, Ec, ec, ecu, Type: str, ax, ID = 0):
02930     """
02931     Function that plots the confined/unconfined Concrete04 stress-strain curve.
02932 
02933     @param fc (float): Compressive concrete yield strength (needs to be negative).
02934     @param Ec (float): Young modulus.
02935     @param ec (float): Compressive concrete yield strain.
02936     @param ecu (float): Compressive concrete failure strain (negative).
02937     @param Type (str): Type of concrete (confined = 'C', unconfined = 'U')
02938     @param ax (matplotlib.axes._subplots.AxesSubplot): The figure's wrapper.
02939     @param ID (int, optional): ID of the material model. Defaults to 0 (= not defined).
02940 
02941     @exception NameError:
02942 
02943     Example: to create the plot, call this line to pass the correct ax:
02944         fig, ax = plt.subplots()
02945     """
02946     if Type == "C":
02947         name = "Confined (Co04)"
02948     elif Type == "U":
```

```
        name = "Unconfined (Co04)"
    else:
        raise NameError("Type should be C or U (ID={})".format(ID))

    # Data for plotting
    N = 1000
    x_axis = np.zeros(N)
    y_axis = np.zeros(N)
    for i in range(N):
        x_axis[i] = i/N*ecu
        y_axis[i] = Concrete04Funct(fc, x_axis[i], ec, Ec)


    ax.plot(x_axis*100.0, y_axis/MPa_unit, 'k-', label = name)
    ax.set(xlabel='Strain [%]', ylabel='Stress [MPa]',
                    title='Mander 1988 (Concrete04) material model (ID={})'.format(ID))
    plt.legend()
    plt.grid()


```

6.10 MemberModel Namespace Reference

Module for the member model.

Classes

- class ElasticElement

    Class that handles the storage and manipulation of a elastic element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.
- class ElasticElementSteelIShape

    Class that is the children of ElasticElement and combine the class SteelIShape (section) to retrieve the information needed.
- class ForceBasedElement

    Class that handles the storage and manipulation of a force-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.
- class ForceBasedElementFibersCircRCCircShape

    Class that is the children of ForceBasedElement and combine the class FibersCircRCCircShape (fiber section) to retrieve the information needed.
- class ForceBasedElementFibersIShapeSteelIShape

    Class that is the children of ForceBasedElement and combine the class FibersIShapeSteelIShape (fiber section) to retrieve the information needed.
- class ForceBasedElementFibersRectRCRectShape

    Class that is the children of ForceBasedElement and combine the class FibersRectRCRectShape (fiber section) to retrieve the information needed.
- class GIFBElement

    Class that handles the storage and manipulation of a Gradient-Inelastic Flexibility-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.
- class GIFBElementFibersCircRCCircShape

    Class that is the children of GIFBElement and combine the class FibersCircRCCircShape (fiber section) to retrieve the information needed.
- class GIFBElementFibersRectRCRectShape

    Class that is the children of GIFBElement and combine the class FibersRectRCRectShape (fiber section) to retrieve the information needed.
- class GIFBElementRCCircShape

    Class that is the children of GIFBElement and combine the class RCCircShape (section) to retrieve the information needed.
- class GIFBElementRCRectShape

Class that is the children of GIFBElement and combine the class RCRectShape (section) to retrieve the information needed.

- class MemberModel

  Parent abstract class for the storage and manipulation of a member's information (mechanical and geometrical parameters, etc) and the initialisation in the model.
- class PanelZone

  Class that handles the storage and manipulation of a panel zone's information (mechanical and geometrical parameters, etc) and the initialisation in the model.
- class PanelZoneRCS

  WIP: Class that is the children of PanelZone and it's used for the panel zone in a RCS (RC column continous, Steel beam).
- class PanelZoneSteelIShape

  Class that is the children of PanelZone and combine the class SteelIShape (section) to retrieve the information needed.
- class PanelZoneSteelIShapeGupta1999

  Class that is the children of PanelZoneSteelIShape and automatically create the spring material model Gupta 1999 (ID = master_node_ID).
- class PanelZoneSteelIShapeSkiadopoulos2021

  Class that is the children of PanelZoneSteelIShape and automatically create the spring material model Skiadopoulos 2021 (ID = master_node_ID).
- class SpringBasedElement

  Class that handles the storage and manipulation of a spring-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.
- class SpringBasedElementModifiedIMKSteelIShape

  Class that is the children of SpringBasedElement and combine the class SteelIShape (section) to retrieve the information needed.
- class SpringBasedElementSteelIShape

  Class that is the children of SpringBasedElement and combine the class SteelIShape (section) to retrieve the information needed.

Functions

- def DefinePanelZoneElements (MasterNodeID, E, RigidA, RigidI, TransfID)

  Function that defines the 8 panel zone elements.
- def DefinePanelZoneNodes (int MasterNodeID, MidPanelZoneWidth, MidPanelZoneHeight)

  Function that defines the remaining 10 nodes of a panel zone given the dimensions and the master node (top center one).

6.10.1 Detailed Description

Module for the member model.

Carmine Schipani, 2021

6.10.2 Function Documentation

6.10.2.1 DefinePanelZoneElements()

```
def MemberModel.DefinePanelZoneElements (
            MasterNodeID,
            E,
            RigidA,
            RigidI,
            TransfID )
```

Function that defines the 8 panel zone elements.

For the ID convention, see DefinePanelZoneNodes.

Parameters

MasterNodeID	(int): ID of the master node (central top node that should be a grid node).
E	(float): Young modulus.
RigidA	(float): A very rigid area.
RigidI	(float): A very rigid moment of inertia.
TransfID	(int): The geometric transformation (for more information, see OpenSeesPy documentation).

Returns

list: List of lists, wth each list containing the ID of the element, of node i and node j.

Definition at line 432 of file MemberModel.py.

```
00432 def DefinePanelZoneElements(MasterNodeID, E, RigidA, RigidI, TransfID):
00433     """
00434     Function that defines the 8 panel zone elements. For the ID convention, see DefinePanelZoneNodes.
00435 
00436     @param MasterNodeID (int): ID of the master node (central top node that should be a grid node).
00437     @param E (float): Young modulus.
00438     @param RigidA (float): A very rigid area.
00439     @param RigidI (float): A very rigid moment of inertia.
00440     @param TransfID (int): The geometric transformation (for more information, see OpenSeesPy
      documentation).
00441 
00442     @returns list: List of lists, wth each list containing the ID of the element, of node i and node
      j.
00443     """
00444     # Compute the ID of the nodes obeying to the convention used
00445     xy = MasterNodeID
00446     xy1 = IDConvention(xy, 1)
00447     xy01 = IDConvention(xy, 1, 1)
00448     xy02 = IDConvention(xy, 2, 1)
00449     xy03 = IDConvention(xy, 3, 1)
00450     xy04 = IDConvention(xy, 4, 1)
00451     xy05 = IDConvention(xy, 5, 1)
00452     xy06 = IDConvention(xy, 6, 1)
00453     xy07 = IDConvention(xy, 7, 1)
00454     xy08 = IDConvention(xy, 8, 1)
00455     xy09 = IDConvention(xy, 9, 1)
00456     xy10 = IDConvention(xy, 10)
00457 
00458     # Create element IDs using the convention: xy(a)xy(a)   with xy(a) = NodeID i and j
00459     #   Starting at MasterNodeID, clockwise
00460     # if MasterNodeID > 99:
00461     #     print("Warning, convention: MasterNodeID's digits should be 2")
00462 
00463     ele1 = IDConvention(xy, xy1)
00464     ele2 = IDConvention(xy01, xy02)
00465     ele3 = IDConvention(xy02, xy03)
00466     ele4 = IDConvention(xy04, xy05)
00467     ele5 = IDConvention(xy05, xy06)
00468     ele6 = IDConvention(xy07, xy08)
00469     ele7 = IDConvention(xy08, xy09)
00470     ele8 = IDConvention(xy10, xy)
00471 
00472     # Create panel zone elements
```

```
#                               ID    ndI    ndJ       A     E      I      Transf
element("elasticBeamColumn", ele1, xy,    xy1,  RigidA, E, RigidI, TransfID)
element("elasticBeamColumn", ele2, xy01, xy02, RigidA, E, RigidI, TransfID)
element("elasticBeamColumn", ele3, xy02, xy03, RigidA, E, RigidI, TransfID)
element("elasticBeamColumn", ele4, xy04, xy05, RigidA, E, RigidI, TransfID)
element("elasticBeamColumn", ele5, xy05, xy06, RigidA, E, RigidI, TransfID)
element("elasticBeamColumn", ele6, xy07, xy08, RigidA, E, RigidI, TransfID)
element("elasticBeamColumn", ele7, xy08, xy09, RigidA, E, RigidI, TransfID)
element("elasticBeamColumn", ele8, xy10, xy, RigidA, E, RigidI, TransfID)

# Create element array for forther manipulations
element_array = [[ele1, xy, xy1],
    [ele2, xy01, xy02],
    [ele3, xy02, xy03],
    [ele4, xy04, xy05],
    [ele5, xy05, xy06],
    [ele6, xy07, xy08],
    [ele7, xy08, xy09],
    [ele8, xy10, xy]]

return element_array


```

6.10.2.2 DefinePanelZoneNodes()

```
def MemberModel.DefinePanelZoneNodes (
              int MasterNodeID,
               MidPanelZoneWidth,
               MidPanelZoneHeight )
```

Function that defines the remaining 10 nodes of a panel zone given the dimensions and the master node (top center one).

ID convention for the panel zone:
PZNodeID: 12 nodes: top right 1xy (master), 1xy1 top right, 1xy09,1xy10 1xy 1xy1,1xy01

clockwise 10 nodes xy01-xy10 (with double node at corners) o---------—o---------—o
Spring at node 1xy1 | |
PZElemeneID: 8 elements: starting at node 1xy, clockwise | |
(see function DefinePanelZoneElements for more info) | |
| |
1xy08 o o 1xy02
| |
| |
| |
| |
o---------—o---------—o
1xy06,1xy07 1xy05 1xy03,1xy04
Note that the top right node is defined differently because is where the spring is.

Parameters

MasterNodeID	(int): ID of the master node (central top node that should be a grid node).
MidPanelZoneWidth	(float): Mid panel zone width.
MidPanelZoneHeight	(float): Mid panel zone height.

Definition at line 388 of file MemberModel.py.

```
00388 def DefinePanelZoneNodes(MasterNodeID: int, MidPanelZoneWidth, MidPanelZoneHeight):
00389     """
```

```
    Function that defines the remaining 10 nodes of a panel zone given the dimensions and the master node (top center one).
    ID convention for the panel zone: \n
            PZNodeID:        12 nodes: top right 1xy (master), 1xy1 top right, 1xy09,1xy10      1xy      1xy1,1xy01 \n
                             clockwise 10 nodes xy01-xy10 (with double node at corners) o-----------o-----------o       \n
                             Spring at node 1xy1                                                        |         |       \n
            PZElemeneID:     8 elements: starting at node 1xy, clockwise                                |         |       \n
                             (see function DefinePanelZoneElements for more info)                       |         |       \n
                                                                                                        |         |       \n
                                                                                                   1xy08 o        o 1xy02 \n
                                                                                                        |         |       \n
                                                                                                        |         |       \n
                                                                                                        |         |       \n
                                                                                                        |         |       \n
    o-----------o-----------o       \n
    1xy06,1xy07    1xy05    1xy03,1xy04 \n
        Note that the top right node is defined differently because is where the spring is.

    @param MasterNodeID (int): ID of the master node (central top node that should be a grid node).
    @param MidPanelZoneWidth (float): Mid panel zone width.
    @param MidPanelZoneHeight (float): Mid panel zone height.

    """
    # Get node coord and define useful variables
    m_node = np.array(nodeCoord(MasterNodeID))
    AxisCL = m_node[0]
    FloorCL = m_node[1] - MidPanelZoneHeight

    # Convention: Node of the spring (top right) is xy1
    node(MasterNodeID, 1), AxisCL+MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
    # Convention: Two notes in the corners (already defined one, xy1) clockwise from xy01 to xy10
    node(IDConvention(MasterNodeID, 1, 1), AxisCL+MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
    node(IDConvention(MasterNodeID, 2, 1), AxisCL+MidPanelZoneWidth, FloorCL)
    node(IDConvention(MasterNodeID, 3, 1), AxisCL+MidPanelZoneWidth, FloorCL-MidPanelZoneHeight)
    node(IDConvention(MasterNodeID, 4, 1), AxisCL+MidPanelZoneWidth, FloorCL-MidPanelZoneHeight)
    node(IDConvention(MasterNodeID, 5, 1), AxisCL, FloorCL-MidPanelZoneHeight)
    node(IDConvention(MasterNodeID, 6, 1), AxisCL-MidPanelZoneWidth, FloorCL-MidPanelZoneHeight)
    node(IDConvention(MasterNodeID, 7, 1), AxisCL-MidPanelZoneWidth, FloorCL-MidPanelZoneHeight)
    node(IDConvention(MasterNodeID, 8, 1), AxisCL-MidPanelZoneWidth, FloorCL)
    node(IDConvention(MasterNodeID, 9, 1), AxisCL-MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
    node(IDConvention(MasterNodeID, 10), AxisCL-MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
```

6.11 Section Namespace Reference

Module for the section (steel I shape profiles, RC circular/square/rectangular sections).

Classes

- class RCCircShape

  Class that stores funcions, geometric and mechanical properties of RC circular shape profile.
- class RCRectShape

  Class that stores funcions, geometric and mechanical properties of RC rectangular shape profile.
- class RCSquareShape

  Class that is the children of RCRectShape and cover the specific case of square RC sections.
- class Section

  Parent abstract class for the storage and manipulation of a section's information (mechanical and geometrical parameters, etc).
- class SteelIShape

  Class that stores funcions, geometric and mechanical properties of a steel double symmetric I-shape profile.

Functions

- def ComputeACircle (D)

    Function that computes the area of one circle (reinforcing bar or hoop).
- def ComputeRho (A, nr, A_tot)

    Compute the ratio of area of a reinforcement to area of a section.

6.11.1 Detailed Description

Module for the section (steel I shape profiles, RC circular/square/rectangular sections).

Carmine Schipani, 2021

6.11.2 Function Documentation

6.11.2.1 ComputeACircle()

```
def Section.ComputeACircle (
              D )
```

Function that computes the area of one circle (reinforcing bar or hoop).

Parameters

D	(float): Diameter of the circle (reinforcing bar of hoop).

Returns

float: Area the circle (for reinforcing bars or hoops).

Definition at line 763 of file Section.py.

```
def ComputeACircle(D):
    """
    Function that computes the area of one circle (reinforcing bar or hoop).

    @param D (float): Diameter of the circle (reinforcing bar of hoop).

    @returns float: Area the circle (for reinforcing bars or hoops).
    """
    return D**2/4.0*math.pi


```

6.11.2.2 ComputeRho()

```
def Section.ComputeRho (
              A,
              nr,
              A_tot )
```

Compute the ratio of area of a reinforcement to area of a section.

Parameters

A	(float): Area of reinforcement.
nr	(float): Number of reinforcement (allow float for computing ratio with different area; just convert the other areas to one and compute the equivalent n).
A_tot	(float): Area of the concrete.

Returns

float: Ratio.

Definition at line 774 of file Section.py.

```
00774 def ComputeRho(A, nr, A_tot):
00775     """
00776     Compute the ratio of area of a reinforcement to area of a section.
00777
00778     @param A (float): Area of reinforcement.
00779     @param nr (float): Number of reinforcement (allow float for computing ratio with different area;
00780         just convert the other areas to one and compute the equivalent n).
00781     @param A_tot (float): Area of the concrete.
00782
00783     @returns float: Ratio.
00784     """
00785     return nr * A / A_tot
```

6.12 Units Namespace Reference

Module with the units conversion and the definition of the units used as default (m, N, s).

Variables

- float cm2_unit = cm_unit∗cm_unit
- float cm3_unit = cm_unit∗cm_unit∗cm_unit
- float cm4_unit = cm3_unit∗cm_unit
- float cm_unit = m_unit∗1e-2
- float dm2_unit = dm_unit∗dm_unit
- float dm3_unit = dm_unit∗dm_unit∗dm_unit
- float dm4_unit = dm3_unit∗dm_unit
- float dm_unit = m_unit∗1e-1
- string force_unit = "N"
- float ft2_unit = ft_unit∗ft_unit
- float ft3_unit = ft_unit∗ft_unit∗ft_unit
- float ft4_unit = ft3_unit∗ft_unit
- float ft_unit = m_unit∗0.3048
- float GN_unit = N_unit∗1e9
- float GPa_unit = Pa_unit∗1e9
- float hours_unit = min_unit∗60
- float inch2_unit = inch_unit∗inch_unit
- float inch3_unit = inch_unit∗inch_unit∗inch_unit
- float inch4_unit = inch3_unit∗inch_unit
- float inch_unit = m_unit∗0.0254
- float kg_unit = N_unit∗s_unit∗∗2/m_unit
- float kip_unit = N_unit∗4448.2216
- float km_unit = m_unit∗1e3

- float kN_unit = N_unit∗1e3
- float kNm_unit = kN_unit∗m_unit
- float kNmm_unit = kN_unit∗mm_unit
- float kPa_unit = Pa_unit∗1e3
- float ksi_unit = psi_unit∗1000
- string length_unit = "m"
- float m2_unit = m_unit∗m_unit
- float m3_unit = m_unit∗m_unit∗m_unit
- float m4_unit = m3_unit∗m_unit
- float m_unit = 1.0
- float mile_unit = m_unit∗1609.34
- float min_unit = s_unit∗60
- float mm2_unit = mm_unit∗mm_unit
- float mm3_unit = mm_unit∗mm_unit∗mm_unit
- float mm4_unit = mm3_unit∗mm_unit
- float mm_unit = m_unit∗1e-3
- float MN_unit = N_unit∗1e6
- float MNm_unit = MN_unit∗m_unit
- float MNmm_unit = MN_unit∗mm_unit
- float MPa_unit = Pa_unit∗1e6
- float N_unit = 1.0
- float Nm_unit = N_unit∗m_unit
- float Nmm_unit = N_unit∗mm_unit
- float Pa_unit = N_unit/m2_unit
- float pound_unit = kg_unit∗0.45359237
- float psi_unit = Pa_unit∗6894.76
- float s_unit = 1.0
- float t_unit = kg_unit∗1e3
- string time_unit = "s"

6.12.1 Detailed Description

Module with the units conversion and the definition of the units used as default (m, N, s).

Note that the decision of which unit for each measure (distance, force, mass, time) is equal to 1 is not arbitrary: for example the natural frequency is computed behind the scene by the OpenSeesPy framework, thus the stiffness of the structure divided by the mass should result in a unit of 1 (thus seconds).
Furthermore, there are constants like the gravitational one g that is dependent on this decision. If the units are used in a consistent way (using this library), these issues can be avoided.
Carmine Schipani, 2021

6.12.2 Variable Documentation

6.12.2.1 cm2_unit

```
float cm2_unit = cm_unit∗cm_unit
```

Definition at line 28 of file Units.py.

6.12.2.2 cm3_unit

```
float cm3_unit = cm_unit*cm_unit*cm_unit
```

Definition at line 36 of file Units.py.

6.12.2.3 cm4_unit

```
float cm4_unit = cm3_unit*cm_unit
```

Definition at line 44 of file Units.py.

6.12.2.4 cm_unit

```
float cm_unit = m_unit*1e-2
```

Definition at line 19 of file Units.py.

6.12.2.5 dm2_unit

```
float dm2_unit = dm_unit*dm_unit
```

Definition at line 29 of file Units.py.

6.12.2.6 dm3_unit

```
float dm3_unit = dm_unit*dm_unit*dm_unit
```

Definition at line 37 of file Units.py.

6.12.2.7 dm4_unit

```
float dm4_unit = dm3_unit*dm_unit
```

Definition at line 45 of file Units.py.

6.12.2.8 dm_unit

```
float dm_unit = m_unit*1e-1
```

Definition at line 20 of file Units.py.

6.12.2.9 force_unit

```
string force_unit = "N"
```

Definition at line 13 of file Units.py.

6.12.2.10 ft2_unit

```
float ft2_unit = ft_unit*ft_unit
```

Definition at line 32 of file Units.py.

6.12.2.11 ft3_unit

```
float ft3_unit = ft_unit*ft_unit*ft_unit
```

Definition at line 40 of file Units.py.

6.12.2.12 ft4_unit

```
float ft4_unit = ft3_unit*ft_unit
```

Definition at line 48 of file Units.py.

6.12.2.13 ft_unit

```
float ft_unit = m_unit*0.3048
```

Definition at line 23 of file Units.py.

6.12.2.14 GN_unit

```
float GN_unit = N_unit*1e9
```

Definition at line 53 of file Units.py.

6.12.2.15 GPa_unit

```
float GPa_unit = Pa_unit*1e9
```

Definition at line 73 of file Units.py.

6.12.2.16 hours_unit

```
float hours_unit = min_unit*60
```

Definition at line 79 of file Units.py.

6.12.2.17 inch2_unit

```
float inch2_unit = inch_unit*inch_unit
```

Definition at line 31 of file Units.py.

6.12.2.18 inch3_unit

```
float inch3_unit = inch_unit*inch_unit*inch_unit
```

Definition at line 39 of file Units.py.

6.12.2.19 inch4_unit

```
float inch4_unit = inch3_unit*inch_unit
```

Definition at line 47 of file Units.py.

6.12.2.20 inch_unit

```
float inch_unit = m_unit*0.0254
```

Definition at line 22 of file Units.py.

6.12.2.21 kg_unit

```
float kg_unit = N_unit*s_unit**2/m_unit
```

Definition at line 65 of file Units.py.

6.12.2.22 kip_unit

```
float kip_unit = N_unit*4448.2216
```

Definition at line 54 of file Units.py.

6.12.2.23 km_unit

```
float km_unit = m_unit*1e3
```

Definition at line 21 of file Units.py.

6.12.2.24 kN_unit

```
float kN_unit = N_unit*1e3
```

Definition at line 51 of file Units.py.

6.12.2.25 kNm_unit

```
float kNm_unit = kN_unit*m_unit
```

Definition at line 58 of file Units.py.

6.12.2.26 kNmm_unit

```
float kNmm_unit = kN_unit*mm_unit
```

Definition at line 61 of file Units.py.

6.12.2.27 kPa_unit

```
float kPa_unit = Pa_unit*1e3
```

Definition at line 71 of file Units.py.

6.12.2.28 ksi_unit

```
float ksi_unit = psi_unit*1000
```

Definition at line 75 of file Units.py.

6.12.2.29 length_unit

```
string length_unit = "m"
```

Definition at line 11 of file Units.py.

6.12.2.30 m2_unit

```
float m2_unit = m_unit*m_unit
```

Definition at line 30 of file Units.py.

6.12.2.31 m3_unit

```
float m3_unit = m_unit*m_unit*m_unit
```

Definition at line 38 of file Units.py.

6.12.2.32 m4_unit

```
float m4_unit = m3_unit*m_unit
```

Definition at line 46 of file Units.py.

6.12.2.33 m_unit

```
float m_unit = 1.0
```

Definition at line 10 of file Units.py.

6.12.2.34 mile_unit

```
float mile_unit = m_unit*1609.34
```

Definition at line 24 of file Units.py.

6.12.2.35 min_unit

```
float min_unit = s_unit*60
```

Definition at line 78 of file Units.py.

6.12.2.36 mm2_unit

```
float mm2_unit = mm_unit*mm_unit
```

Definition at line 27 of file Units.py.

6.12.2.37 mm3_unit

```
float mm3_unit = mm_unit*mm_unit*mm_unit
```

Definition at line 35 of file Units.py.

6.12.2.38 mm4_unit

```
float mm4_unit = mm3_unit*mm_unit
```

Definition at line 43 of file Units.py.

6.12.2.39 mm_unit

```
float mm_unit = m_unit*1e-3
```

Definition at line 18 of file Units.py.

6.12.2.40 MN_unit

```
float MN_unit = N_unit*1e6
```

Definition at line 52 of file Units.py.

6.12.2.41 MNm_unit

```
float MNm_unit = MN_unit*m_unit
```

Definition at line 59 of file Units.py.

6.12.2.42 MNmm_unit

```
float MNmm_unit = MN_unit*mm_unit
```

Definition at line 62 of file Units.py.

6.12.2.43 MPa_unit

```
float MPa_unit = Pa_unit*1e6
```

Definition at line 72 of file Units.py.

6.12.2.44 N_unit

```
float N_unit = 1.0
```

Definition at line 12 of file Units.py.

6.12.2.45 Nm_unit

```
float Nm_unit = N_unit*m_unit
```

Definition at line 57 of file Units.py.

6.12.2.46 Nmm_unit

```
float Nmm_unit = N_unit*mm_unit
```

Definition at line 60 of file Units.py.

6.12.2.47 Pa_unit

```
float Pa_unit = N_unit/m2_unit
```

Definition at line 70 of file Units.py.

6.12.2.48 pound_unit

```
float pound_unit = kg_unit*0.45359237
```

Definition at line 67 of file Units.py.

6.12.2.49 psi_unit

```
float psi_unit = Pa_unit*6894.76
```

Definition at line 74 of file Units.py.

6.12.2.50 s_unit

```
float s_unit = 1.0
```

Definition at line 14 of file Units.py.

6.12.2.51 t_unit

```
float t_unit = kg_unit*1e3
```

Definition at line 66 of file Units.py.

6.12.2.52 time_unit

```
string time_unit = "s"
```

Definition at line 15 of file Units.py.

Chapter 7

Class Documentation

7.1 Analysis Class Reference

Class dedicated to the analysis of the OpenSeesPy model.

Public Member Functions

- def __init__ (self, str data_dir, str name_ODB, algo="KrylovNewton", test_type="NormDispIncr", test_opt=0, max_iter=MAX_ITER, tol=TOL, allow_smaller_step=False)

  The constructor of the class.
- def DeformedShape (self, scale=1, animate=False, dt=0.01)

  Method that shows the final deformed shape of the model.
- def FiberResponse (self, int ele_fiber_ID_analysed, fiber_section=1, animate_stress=False, animate_↩ strain=False, fps=25)

  Method that shows the final stress response of the fiber section chosen.
- def Gravity (self, list loaded_nodes, list Fy, int timeSeries_ID, int pattern_ID, n_step=10, time↩ Series_type="Linear", pattern_type="Plain", constraints_type="Plain", numberer_type="RCM", system_↩ type="BandGeneral", analysis_type="Static", show_plot=False)

  Method to perform the gravity analyisis with vertical loadings (load-control).
- def LateralForce (self, list loaded_nodes, list Fx, int timeSeries_ID, int pattern_ID, n_step=1000, fiber_↩ ID_analysed=-1, fiber_section=1, timeSeries_type="Linear", pattern_type="Plain", constraints_type="Plain", numberer_type="RCM", system_type="BandGeneral", analysis_type="Static", show_plot=True, block=False)

  Method to perform the lateral force analyisis with lateral loading (load-control).
- def LoadingProtocol (self, int CtrlNode, np.ndarray discr_LP, int timeSeries_ID, int pattern_ID, Fx=1 ∗kN_unit, ele_fiber_ID_analysed=-1, fiber_section=1, timeSeries_type="Linear", pattern_type="Plain", constraints↩ _type="Plain", numberer_type="RCM", system_type="UmfPack", analysis_type="Static", show_plot=True, block=False)

  Method to perform a loading protocol analysis (displacement-control).
- def Pushover (self, int CtrlNode, Dmax, Dincr, int timeSeries_ID, int pattern_ID, Fx=1 ∗kN_unit, ele_fiber_↩ ID_analysed=-1, fiber_section=1, timeSeries_type="Linear", pattern_type="Plain", constraints_type="Plain", numberer_type="RCM", system_type="UmfPack", analysis_type="Static", show_plot=True, block=False)

  Method to perform a pushover analysis (displacement-control).

Public Attributes

- algo
- allow_smaller_step
- data_dir
- load_case
- max_iter
- name_ODB
- test_opt
- test_type
- tol

7.1.1 Detailed Description

Class dedicated to the analysis of the OpenSeesPy model.

The Gravity method should be run first to perform the Load-control analysis (apply the vertical load). If no vertical load, this method can be omitted.
Then only one of the Displacement-control (Pushover or LoadingProtocol) or Load-control (LateralForce) analysis can ran.
After the analysis reach convergence in the last step, for the postprocessing, the DeformedShape method can be used to see the final deformed shape and the animation of the entire loading protocol; the FiberResponse method can be used to see the animation of the same fiber section recorded during the analysis (strain and/or stress).

Definition at line 16 of file AnalysisAndPostProcessing.py.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 __init__()

```
def __init__ (
              self,
             str data_dir,
             str name_ODB,
              algo = "KrylovNewton",
              test_type = "NormDispIncr",
              test_opt = 0,
              max_iter = MAX_ITER,
              tol = TOL,
              allow_smaller_step = False )
```

The constructor of the class.

Parameters

data_dir	(str): Directory in which the results from the analysis will be stored. Use the recorders (from OpenSeesPy) or the Record method from MemberModel.
name_ODB	(str): Name for the folder in which the data for the animations and the fibers are stored.

Parameters

algo	(str, optional): Type of alghoritm chosen for the analysis. It detemines how to construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the non-linear equation. For more information on the available types, see the OpenSeesPy documentation. Defaults to "KrylovNewton".
test_type	(str, optional): Type of test chosen for the analysis. It determines how to construct a ConvergenceTest object. Certain SolutionAlgorithm objects require a ConvergenceTest object to determine if convergence has been achieved at the end of an iteration step. For more information on the available types, see the OpenSeesPy documentation. Defaults to "NormDispIncr".
test_opt	(int, optional): Print-flag from 0 to 5 used to receive more info during the iteration (for example: 0 print nothing and 2 print information on norms and number of iterations at end of successful test). For more information, see the OpenSeesPy documentation. Defaults to 0.
max_iter	(float, optional): Maximal number of iterations to check. Defaults to MAX_ITER (from Constants Module).
tol	(float, optional): Tolerance criteria used to check for convergence. Defaults to TOL (from Constants Module).
allow_smaller_step	(bool, optional): Allow smaller steps in the displacement-control analysis. Defaults to False.

Exceptions

NegativeValue	The argument max_iter should be positive.
NegativeValue	The argument tol should be positive.

Definition at line 22 of file AnalysisAndPostProcessing.py.

```
00022     def __init__(self, data_dir: str,  name_ODB: str, algo = "KrylovNewton", test_type = "NormDispIncr", test_opt = 0, max_iter = MAX_ITER, tol = TOL, allow_smaller_step = False):
00023         """
00024         The constructor of the class.
00025 
00026         @param data_dir (str): Directory in which the results from the analysis will be stored. Use the recorders (from OpenSeesPy) or the Record method from MemberModel.
00027         @param name_ODB (str): Name for the folder in which the data for the animations and the fibers are stored.
00028         @param algo (str, optional): Type of alghoritm chosen for the analysis. It detemines how to construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the non-linear equation.
00029             For more information on the available types, see the OpenSeesPy documentation. Defaults to "KrylovNewton".
00030         @param test_type (str, optional): Type of test chosen for the analysis. It determines how to construct a ConvergenceTest object.
00031             Certain SolutionAlgorithm objects require a ConvergenceTest object to determine if convergence has been achieved at the end of an iteration step.
00032             For more information on the available types, see the OpenSeesPy documentation. Defaults to "NormDispIncr".
00033         @param test_opt (int, optional): Print-flag from 0 to 5 used to receive more info during the iteration
00034             (for example: 0 print nothing and 2 print information on norms and number of iterations at end of successful test).
00035             For more information, see the OpenSeesPy documentation. Defaults to 0.
00036         @param max_iter (float, optional): Maximal number of iterations to check. Defaults to MAX_ITER (from Constants Module).
00037         @param tol (float, optional): Tolerance criteria used to check for convergence. Defaults to TOL (from Constants Module).
00038         @param allow_smaller_step (bool, optional): Allow smaller steps in the displacement-control analysis. Defaults to False.
00039 
00040         @exception NegativeValue: The argument max_iter should be positive.
00041         @exception NegativeValue: The argument tol should be positive.
00042         """
00043         if max_iter < 0: raise NegativeValue()
00044         if tol < 0: raise NegativeValue()
00045         if not os.path.exists(data_dir):
00046             print("Folder {} not found in this directory; creating one".format(data_dir))
00047             os.makedirs(data_dir)
```

```
        self.data_dir = data_dir
        self.name_ODB = name_ODB
        self.algo = algo
        self.test_type = test_type
        self.tol = tol
        self.test_opt = test_opt
        self.max_iter = max_iter
        self.allow_smaller_step = allow_smaller_step
        self.load_case = "None"

```

7.1.3 Member Function Documentation

7.1.3.1 DeformedShape()

```
def DeformedShape (
                self,
                scale = 1,
                animate = False,
                dt = 0.01 )
```

Method that shows the final deformed shape of the model.

It can also show the animation that shows how the model behaved during the analysis.

Parameters

scale	(int, optional): The scaling factor to magnify the deformation. The value should be adjusted for each model. Defaults to 1.
animate	(bool, optional): Option to show the animation of the model during the analysis. Defaults to False.
dt	(float, optional): The time step between every iteration. Defaults to 0.01.

Exceptions

NameError	The methods for the analysis were not called.

Definition at line 652 of file AnalysisAndPostProcessing.py.

```
00652     def DeformedShape(self, scale = 1, animate = False, dt = 0.01):
00653         """
00654         Method that shows the final deformed shape of the model. It can also show the animation that
      shows how the model behaved during the analysis.
00655
00656         @param scale (int, optional): The scaling factor to magnify the deformation. The value should
      be adjusted for each model. Defaults to 1.
00657         @param animate (bool, optional): Option to show the animation of the model during the
      analysis. Defaults to False.
00658         @param dt (float, optional): The time step between every iteration. Defaults to 0.01.
00659
00660         @exception NameError: The methods for the analysis were not called.
00661         """
00662         if self.load_case == "None": raise NameError("The analysis is not complete.")
00663
00664         # Display deformed shape, the scaling factor needs to be adjusted for each model
00665         opsplt.plot_deformedshape(Model = self.name_ODB, LoadCase=self.load_case, scale = scale)
00666         if animate:
00667             opsplt.animate_deformedshape(Model = self.name_ODB, LoadCase=self.load_case, dt = dt,
      scale = scale)
00668
```

```
00669
```

7.1.3.2 FiberResponse()

```
def FiberResponse (
        self,
        int ele_fiber_ID_analysed,
        fiber_section = 1,
        animate_stress = False,
        animate_strain = False,
        fps = 25 )
```

Method that shows the final stress response of the fiber section chosen.

It can also show the animation that shows how the fiber section behaved during the analysis. The fiber ID and section needs to be recorded during the analysis, thus if the method LateralForce, Pushover or LoadingProtocol was used, the same fiber ID and section need to be used.

Parameters

ele_fiber_ID_analysed	(int): The ID of the analysed fiber. If fibers are present in the model and the user wants to save ODB data (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the fiber chosen. -1 will ignore the storage of data for fibers.
fiber_section	(int, optional): The section number, i.e. the Gauss integratio number. If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
animate_stress	(bool, optional): Option to show the animation of the fiber stress during the analysis. Defaults to False.
animate_strain	(bool, optional): Option to show the animation of the fiber strain during the analysis. Defaults to False.
fps	(int, optional): Number of frame per seconds for the animations. Defaults to 25.

Exceptions

NameError	The methods for the analysis were not called.

Definition at line 670 of file AnalysisAndPostProcessing.py.

```
00670     def FiberResponse(self, ele_fiber_ID_analysed: int, fiber_section = 1, animate_stress = False, animate_strain = False, fps = 25):
00671         """
00672         Method that shows the final stress response of the fiber section chosen.
00673         It can also show the animation that shows how the fiber section behaved during the analysis. The fiber ID and section needs to be recorded during the analysis,
00674         thus if the method LateralForce, Pushover or LoadingProtocol was used, the same fiber ID and section need to be used.
00675 
00676         @param ele_fiber_ID_analysed (int): The ID of the analysed fiber. If fibers are present in the model and the user wants to save ODB data
00677             (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the fiber chosen.
00678             -1 will ignore the storage of data for fibers.
00679         @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
00680             If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
00681         @param animate_stress (bool, optional): Option to show the animation of the fiber stress during the analysis. Defaults to False.
00682         @param animate_strain (bool, optional): Option to show the animation of the fiber strain during the analysis. Defaults to False.
00683         @param fps (int, optional): Number of frame per seconds for the animations. Defaults to 25.
```

```
        @exception NameError: The methods for the analysis were not called.
        """
        if self.load_case == "None": raise NameError("The analysis is not complete.")

        opsplt.plot_fiberResponse2D(self.name_ODB, self.load_case, ele_fiber_ID_analysed, fiber_section, InputType = 'stress')
        if animate_stress:
            ani1 = opsplt.animate_fiberResponse2D(self.name_ODB, self.load_case, ele_fiber_ID_analysed, fiber_section, InputType = 'stress', fps = fps)
        if animate_strain:
            ani1 = opsplt.animate_fiberResponse2D(self.name_ODB, self.load_case, ele_fiber_ID_analysed, fiber_section, InputType = 'strain', fps = fps)


```

7.1.3.3 Gravity()

```
def Gravity (
              self,
             list loaded_nodes,
             list Fy,
             int timeSeries_ID,
             int pattern_ID,
              n_step = 10,
              timeSeries_type = "Linear",
              pattern_type = "Plain",
              constraints_type = "Plain",
              numberer_type = "RCM",
              system_type = "BandGeneral",
              analysis_type = "Static",
              show_plot = False )
```

Method to perform the gravity analiysis with vertical loadings (load-control).

It can be used before calling the Pushover or LoadingProtocol methods that perform the actual anlysis. If no vertical loadings present, this method can be avoided.

Parameters

loaded_nodes	(list): List of nodes that are loaded by the the forces in Fy. The first node will be recorded (thus usually should be in the roof).
Fy	(list): List of vertical loadings (negative is toward the ground, thus compression; see global coordinate system).
timeSeries_ID	(int): ID of the timeseries.
pattern_ID	(int): ID of the pattern.
n_step	(int, optional): Number of steps used to during the analysis to reach the objective state (with 100% vertical loadings imposed). Defaults to 10.
timeSeries_type	(str, optional): Type of timeseries chosen. For more information, see the OpenSeesPy documentation. Defaults to "Linear".
pattern_type	(str, optional): Type of pattern chosen. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
constraints_type	(str, optional): Type of contraints chosen. It detemines how the constraint equations are enforced in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
numberer_type	(str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom. For more information, see the OpenSeesPy documentation. Defaults to "RCM".

Parameters

system_type	(str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "BandGeneral".
analysis_type	(str, optional): Type of analysis chosen. It determines how to construct the Analysis object, which defines what type of analysis is to be performed. For more information, see the OpenSeesPy documentation. Defaults to "Static".
show_plot	(bool, optional): Option to show the 'vertical displacement vs. vertical loading' curve after the analysis. Defaults to False.

Exceptions

WrongDimension	The dimension of the loaded_nodes and Fy arguments needs to be the same.
NegativeValue	The ID of timeSeries_ID needs to be a positive integer.
NegativeValue	The ID of pattern_ID needs to be a positive integer.

Definition at line 60 of file AnalysisAndPostProcessing.py.

```
        constraints_type = "Plain", numberer_type = "RCM", system_type = "BandGeneral", analysis_type = "Static", show_plot = False):
        """
        Method to perform the gravity analyisis with vertical loadings (load-control).
        It can be used before calling the Pushover or LoadingProtocol methods that perform the actual anlysis. If no vertical loadings present, this method can be avoided.

        @param loaded_nodes (list): List of nodes that are loaded by the the forces in Fy. The first node will be recorded (thus usually should be in the roof).
        @param Fy (list): List of vertical loadings (negative is toward the ground, thus compression; see global coordinate system).
        @param timeSeries_ID (int): ID of the timeseries.
        @param pattern_ID (int): ID of the pattern.
        @param n_step (int, optional): Number of steps used to during the analysis to reach the objective state (with 100% vertical loadings imposed). Defaults to 10.
        @param timeSeries_type (str, optional): Type of timeseries chosen.
            For more information, see the OpenSeesPy documentation. Defaults to "Linear".
        @param pattern_type (str, optional): Type of pattern chosen.
            For more information, see the OpenSeesPy documentation. Defaults to "Plain".
        @param constraints_type (str, optional): Type of contraints chosen. It detemines how the constraint equations are enforced in the analysis.
            For more information, see the OpenSeesPy documentation. Defaults to "Plain".
        @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom.
            For more information, see the OpenSeesPy documentation. Defaults to "RCM".
        @param system_type (str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis.
            For more information, see the OpenSeesPy documentation. Defaults to "BandGeneral".
        @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct the Analysis object, which defines what type of analysis is to be performed.
            For more information, see the OpenSeesPy documentation. Defaults to "Static".
        @param show_plot (bool, optional): Option to show the 'vertical displacement vs. vertical loading' curve after the analysis. Defaults to False.

        @exception WrongDimension: The dimension of the loaded_nodes and Fy arguments needs to be the same.
        @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
        @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
        """
        if len(loaded_nodes) != len(Fy): raise WrongDimension()
        if timeSeries_ID < 1: raise NegativeValue()
        if pattern_ID < 1: raise NegativeValue()

        # for mass defined: opsplt.createODB(self.name_ODB, "Gravity", Nmodes = nEigen);
        # for tracking gravity with ODB: opsplt.createODB(self.name_ODB, "Gravity");

        # Create load pattern
        timeSeries(timeSeries_type, timeSeries_ID)
        pattern(pattern_type, timeSeries_ID, pattern_ID)
        for ii, node_ID in enumerate(loaded_nodes):
            load(node_ID, 0.0, Fy[ii], 0.0)      # load(IDNode, Fx, Fy, Mz)
        DGravity = 1.0/n_step                    # load increment

```

```
        # Set up analysis options
        constraints(constraints_type)   # how it handles boundary conditions
        numberer(numberer_type)         # renumber dof's to minimize band-width (optimization)
        system(system_type)             # how to store and solve the system of equations in the analysis
                                        # For static model, BandGeneral, for transient and/or big model, UmfPack
        integrator("LoadControl", DGravity) # LoadControl and DisplacementControl only with static model, linear TimeSeries w/ factor of 1
                                            # Newmark used for transient model
        algorithm("Newton")             # placeholder
        analysis(analysis_type)         # define type of analysis: static for pushover

        # Analysis
        dataG = np.zeros((n_step+1,2))
        print("")
        print("Gravity analysis starts")
        for iteration in range(n_step):
            convergence = self.__LoadCtrlLoop(DGravity, iteration,
                self.algo, self.test_type, self.tol, self.test_opt, self.max_iter)
            if convergence != 0: break
            dataG[iteration+1,0] = nodeDisp(loaded_nodes[0], 2)/mm_unit
            dataG[iteration+1,1] = getLoadFactor(pattern_ID)*Fy[0]/kN_unit

        if show_plot:
            plt.plot(dataG[:,0], dataG[:,1])
            plt.xlabel('Vertical Displacement [mm]')
            plt.ylabel('Vertical Load [kN]')
            plt.title('Gravity curve')
            plt.show()

        loadConst("-time", 0.0)

        print("")
        print("Gravity complete")


```

7.1.3.4 LateralForce()

```
def LateralForce (
        self,
        list loaded_nodes,
        list Fx,
        int timeSeries_ID,
        int pattern_ID,
        n_step = 1000,
        fiber_ID_analysed = -1,
        fiber_section = 1,
        timeSeries_type = "Linear",
        pattern_type = "Plain",
        constraints_type = "Plain",
        numberer_type = "RCM",
        system_type = "BandGeneral",
        analysis_type = "Static",
        show_plot = True,
        block = False )
```

Method to perform the lateral force analysis with lateral loading (load-control).

If this method is called, the LoadingProtocol and Pushover methods should be avoided.

Parameters

loaded_nodes	(list): List of nodes that are loaded by the the forces in Fx. The first node will be recorded (thus usually should be in the roof).
Fx	(list): List of horizontal loadings (negative is toward left; see global coordinate system).

Parameters

timeSeries_ID	(int): ID of the timeseries.
pattern_ID	(int): ID of the pattern.
n_step	(int, optional): Number of steps used to during the analysis to reach the objective state (with 100% horizontal loadings imposed). Defaults to 1000.
fiber_ID_analysed	(int, optional): The ID of the analysed fiber. If fibers are present in the model and the user wants to save ODB data (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the fiber chosen. -1 will ignore the storage of data for fibers. Defaults to -1.
fiber_section	(int, optional): The section number, i.e. the Gauss integratio number. If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
timeSeries_type	(str, optional): Type of timeseries chosen. For more information, see the OpenSeesPy documentation. Defaults to "Linear".
pattern_type	(str, optional): Type of pattern chosen. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
constraints_type	(str, optional): Type of contraints chosen. It detemines how the constraint equations are enforced in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
numberer_type	(str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom. For more information, see the OpenSeesPy documentation. Defaults to "RCM".
system_type	(str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "BandGeneral".
analysis_type	(str, optional): Type of analysis chosen. It determines how to construct the Analysis object, which defines what type of analysis is to be performed. For more information, see the OpenSeesPy documentation. Defaults to "Static".
show_plot	(bool, optional): Option to show the 'Horizontal displacement vs. Horizontal loading' curve after the analysis. Defaults to True.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Exceptions

WrongDimension	The dimension of the loaded_nodes and Fx arguments needs to be the same.
NegativeValue	The ID of timeSeries_ID needs to be a positive integer.
NegativeValue	The ID of pattern_ID needs to be a positive integer.
NegativeValue	The ID of fiber_ID_analysed needs to be a positive integer.

Definition at line 137 of file AnalysisAndPostProcessing.py.

```
        show_plot = True, block = False):
        """
        Method to perform the lateral force analyisis with lateral loading (load-control).
        If this method is called, the LoadingProtocol and Pushover methods should be avoided.

        @param loaded_nodes (list): List of nodes that are loaded by the the forces in Fx. The first
    node will be recorded (thus usually should be in the roof).
        @param Fx (list): List of horizontal loadings (negative is toward left; see global coordinate
    system).
        @param timeSeries_ID (int): ID of the timeseries.
        @param pattern_ID (int): ID of the pattern.
        @param n_step (int, optional): Number of steps used to during the analysis to reach the
    objective state (with 100% horizontal loadings imposed). Defaults to 1000.
        @param fiber_ID_analysed (int, optional): The ID of the analysed fiber. If fibers are present
    in the model and the user wants to save ODB data
            (to use in the post-processing with for example FiberResponse), assign to this argument
    the ID of the fiber chosen.
```

```
            -1 will ignore the storage of data for fibers. Defaults to -1.
        @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
            If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
        @param timeSeries_type (str, optional): Type of timeseries chosen.
            For more information, see the OpenSeesPy documentation. Defaults to "Linear".
        @param pattern_type (str, optional): Type of pattern chosen.
            For more information, see the OpenSeesPy documentation. Defaults to "Plain".
        @param constraints_type (str, optional): Type of contraints chosen. It detemines how the constraint equations are enforced in the analysis.
            For more information, see the OpenSeesPy documentation. Defaults to "Plain".
        @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom.
            For more information, see the OpenSeesPy documentation. Defaults to "RCM".
        @param system_type (str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis.
            For more information, see the OpenSeesPy documentation. Defaults to "BandGeneral".
        @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct the Analysis object, which defines what type of analysis is to be performed.
            For more information, see the OpenSeesPy documentation. Defaults to "Static".
        @param show_plot (bool, optional): Option to show the 'Horizontal displacement vs. Horizontal loading' curve after the analysis. Defaults to True.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

        @exception WrongDimension: The dimension of the loaded_nodes and Fx arguments needs to be the same.
        @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
        @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
        @exception NegativeValue: The ID of fiber_ID_analysed needs to be a positive integer.
        """
        if len(loaded_nodes) != len(Fx): raise WrongDimension()
        if timeSeries_ID < 1: raise NegativeValue()
        if pattern_ID < 1: raise NegativeValue()
        if fiber_ID_analysed != -1 and fiber_ID_analysed < 1: raise NegativeValue()

        # for mass defined: opsplt.createODB(self.name_ODB, "LateralForce", Nmodes = nEigen);
        opsplt.createODB(self.name_ODB, "LateralForce");
        if fiber_ID_analysed != -1: opsplt.saveFiberData2D(self.name_ODB, "LateralForce", fiber_ID_analysed, fiber_section)

        # Create load pattern
        timeSeries(timeSeries_type, timeSeries_ID)
        pattern(pattern_type, timeSeries_ID, pattern_ID)
        for ii, node_ID in enumerate(loaded_nodes):
            load(node_ID, Fx[ii], 0.0, 0.0)      # load(IDNode, Fx, Fy, Mz)
        force = 1.0/n_step                      # load increment

        # Set up analysis options
        constraints(constraints_type)   # how it handles boundary conditions
        numberer(numberer_type)         # renumber dof's to minimize band-width (optimization)
        system(system_type)             # how to store and solve the system of equations in the analysis
                                        # For static model, BandGeneral, for transient and/or big model, UmfPack
        integrator("LoadControl", force)# LoadControl and DisplacementControl only with static model, linear TimeSeries w/ factor of 1
                                        # Newmark used for transient model
        algorithm("Newton")             # placeholder
        analysis(analysis_type)         # define type of analysis: static for pushover

        # Analysis
        dataLF = np.zeros((n_step+1,2))
        print("")
        print("Lateral Force analysis starts")
        for iteration in range(n_step):
            convergence = self.__LoadCtrlLoop(force, iteration,
                self.algo, self.test_type, self.tol, self.test_opt, self.max_iter)
            if convergence != 0: break
            dataLF[iteration+1,0] = nodeDisp(loaded_nodes[0], 1)/mm_unit
            dataLF[iteration+1,1] = getLoadFactor(pattern_ID)*Fx[0]/kN_unit

        if show_plot:
            plt.plot(dataLF[:,0], dataLF[:,1])
            plt.xlabel('Lateral Displacement [mm]')
            plt.ylabel('Lateral Load [kN]')
            plt.title('Lateral force curve')
            if block:
                plt.show()

        loadConst("-time", 0.0)

        print("")
        print("Lateral force complete")
        self.load_case = "LateralForce"

        wipe()
```

7.1.3.5 LoadingProtocol()

```
def LoadingProtocol (
              self,
             int CtrlNode,
             np.ndarray discr_LP,
             int timeSeries_ID,
             int pattern_ID,
              Fx = 1*kN_unit,
              ele_fiber_ID_analysed = -1,
              fiber_section = 1,
              timeSeries_type = "Linear",
              pattern_type = "Plain",
              constraints_type = "Plain",
              numberer_type = "RCM",
              system_type = "UmfPack",
              analysis_type = "Static",
              show_plot = True,
              block = False )
```

Method to perform a loading protocol analysis (displacement-control).

If this method is called, the Pushover and LateralForce methods should be avoided.

Parameters

CtrlNode	(int): The node that will be used to impose the displacement from the discr_LP to perform the analysis.
discr_LP	(np.ndarray): The loading protocol array (1 dimension) discretised. It needs to be filled with imposed displacement, not SDR. Use the functions DiscretizeLoadProtocol and DiscretizeLinearly in FunctionalFeatures module to help create and/or discretise one.
timeSeries_ID	(int): ID of the timeseries.
pattern_ID	(int): ID of the pattern.
Fx	(float, optional): The force imposed at the control node CtrlNode. It is used for convergence reasons and it can be arbitrarly small. Defaults to 1∗kN_unit.
ele_fiber_ID_analysed	(int, optional): The ID of the analysed element with fibers. If fibers are present in the model and the user wants to save ODB data (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the element with fibers chosen. -1 will ignore the storage of data for fibers. Defaults to -1.
fiber_section	(int, optional): The section number, i.e. the Gauss integratio number. If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
timeSeries_type	(str, optional): Type of timeseries chosen. For more information, see the OpenSeesPy documentation. Defaults to "Linear".
pattern_type	(str, optional): Type of pattern chosen. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
constraints_type	(str, optional): Type of contraints chosen. It detemines how the constraint equations are enforced in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
numberer_type	(str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom. For more information, see the OpenSeesPy documentation. Defaults to "RCM".

Parameters

system_type	(str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "UmfPack".
analysis_type	(str, optional): Type of analysis chosen. It determines how to construct the Analysis object, which defines what type of analysis is to be performed. For more information, see the OpenSeesPy documentation. Defaults to "Static".
show_plot	(bool, optional): Option to show the 'lateral displacement vs. lateral loading' curve after the analysis. Defaults to True.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Exceptions

NegativeValue	The ID of CtrlNode needs to be a positive integer.
NegativeValue	The ID of timeSeries_ID needs to be a positive integer.
NegativeValue	The ID of pattern_ID needs to be a positive integer.
NegativeValue	The ID of fiber_ID_analysed needs to be a positive integer if is different from -1.

Definition at line 319 of file AnalysisAndPostProcessing.py.

```
00321         show_plot = True, block = False):
00322         """
00323         Method to perform a loading protocol analysis (displacement-control). If this method is called, the Pushover and LateralForce methods should be avoided.
00324 
00325         @param CtrlNode (int): The node that will be used to impose the displacement from the discr_LP to perform the analysis.
00326         @param discr_LP (np.ndarray): The loading protocol array (1 dimension) discretised. It needs to be filled with imposed displacement, not SDR.
00327             Use the functions DiscretizeLoadProtocol and DiscretizeLinearly in FunctionalFeatures module to help create and/or discretise one.
00328         @param timeSeries_ID (int): ID of the timeseries.
00329         @param pattern_ID (int): ID of the pattern.
00330         @param Fx (float, optional): The force imposed at the control node CtrlNode. It is used for convergence reasons and it can be arbitrarly small.
00331             Defaults to 1*kN_unit.
00332         @param ele_fiber_ID_analysed (int, optional): The ID of the analysed element with fibers. If fibers are present in the model and the user wants to save ODB data
00333             (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the element with fibers chosen.
00334             -1 will ignore the storage of data for fibers. Defaults to -1.
00335         @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
00336             If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
00337         @param timeSeries_type (str, optional): Type of timeseries chosen.
00338             For more information, see the OpenSeesPy documentation. Defaults to "Linear".
00339         @param pattern_type (str, optional): Type of pattern chosen.
00340             For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00341         @param constraints_type (str, optional): Type of contraints chosen. It detemines how the constraint equations are enforced in the analysis.
00342             For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00343         @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom.
00344             For more information, see the OpenSeesPy documentation. Defaults to "RCM".
00345         @param system_type (str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis.
00346             For more information, see the OpenSeesPy documentation. Defaults to "UmfPack".
00347         @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct the Analysis object, which defines what type of analysis is to be performed.
00348             For more information, see the OpenSeesPy documentation. Defaults to "Static".
00349         @param show_plot (bool, optional): Option to show the 'lateral displacement vs. lateral loading' curve after the analysis. Defaults to True.
00350         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
00351 
00352         @exception NegativeValue: The ID of CtrlNode needs to be a positive integer.
00353         @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
00354         @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
00355         @exception NegativeValue: The ID of fiber_ID_analysed needs to be a positive integer if is different from -1.
```

```
        """
        if CtrlNode < 1: raise NegativeValue()
        if timeSeries_ID < 1: raise NegativeValue()
        if pattern_ID < 1: raise NegativeValue()
        if ele_fiber_ID_analysed != -1 and ele_fiber_ID_analysed < 1: raise NegativeValue()

        # for mass defined: opsplt.createODB(self.name_ODB, "LoadingProtocol", Nmodes = nEigen);
        opsplt.createODB(self.name_ODB, "LoadingProtocol");
        if ele_fiber_ID_analysed != -1: opsplt.saveFiberData2D(self.name_ODB, "LoadingProtocol",
    ele_fiber_ID_analysed, fiber_section)

        # Create load pattern
        timeSeries(timeSeries_type, timeSeries_ID)
        pattern(pattern_type, timeSeries_ID, pattern_ID)
        load(CtrlNode, Fx, 0.0, 0.0)     # load(IDNode, Fx, Fy, Mz)
        dU_prev = 0
        Nsteps = np.size(discr_LP)       # number of pushover analysis steps

        # Set up analysis options
        constraints(constraints_type)    # how it handles boundary conditions
        numberer(numberer_type)          # renumber dof's to minimize band-width (optimization)
        system(system_type)              # how to store and solve the system of equations in the
    analysis
                                         # For static model, BandGeneral, for transient and/or big
    model, UmfPack
        integrator("LoadControl", 1)     # placeholder
        algorithm("Newton")              # placeholder
        analysis(analysis_type)          # define type of analysis: static for LoadingProtocol

        # Analysis
        dataLP = np.zeros((Nsteps+1,2))
        next_step = 0
        print("")
        print("Loading Protocol analysis starts")
        for iteration in range(Nsteps):
            # Compute displacement usinf the given loading protocol (discretized)
            dU_next = discr_LP[iteration]
            dU = dU_next - dU_prev
            dU_prev = dU_next

            next_step = ProgressingPercentage(Nsteps, iteration, next_step)
            convergence = self.__LatDispCtrlLoop(CtrlNode, dU, iteration,
                self.algo, self.test_type, self.tol, self.test_opt, self.max_iter,
    self.allow_smaller_step)
            if convergence != 0: break
            dataLP[iteration+1,0] = nodeDisp(CtrlNode, 1)/mm_unit
            dataLP[iteration+1,1] = getLoadFactor(pattern_ID)*Fx/kN_unit

        if show_plot:
            plt.plot(dataLP[:,0], dataLP[:,1])
            plt.xlabel('Horizontal Displacement [mm]')
            plt.ylabel('Horizontal Load [kN]')
            plt.title('Loading Protocol curve')
            if block:
                plt.show()

        print("")
        print("Loading Protocol complete")
        self.load_case = "LoadingProtocol"

        wipe()


```

7.1.3.6 Pushover()

```
def Pushover (
        self,
        int CtrlNode,
        Dmax,
        Dincr,
        int timeSeries_ID,
        int pattern_ID,
        Fx = 1*kN_unit,
        ele_fiber_ID_analysed = -1,
```

```
fiber_section = 1,
timeSeries_type = "Linear",
pattern_type = "Plain",
constraints_type = "Plain",
numberer_type = "RCM",
system_type = "UmfPack",
analysis_type = "Static",
show_plot = True,
block = False )
```

Method to perform a pushover analysis (displacement-control).

If this method is called, the LoadingProtocol and LateralForce methods should be avoided.

Parameters

CtrlNode	(int): The node that will be used to impose the displacement Dmax of the pushover analysis. If the show_plot option is True, the curve displayed follows this node.
Dmax	(float): The imposed displacement.
Dincr	(float): The incremental displacement to reach Dmax. To converge, it should be small enough (1000 times smaller of Dmax).
timeSeries_ID	(int): ID of the timeseries.
pattern_ID	(int): ID of the pattern.
Fx	(float, optional): The force imposed at the control node CtrlNode. It is used for convergence reasons and it can be arbitrarly small. Defaults to 1∗kN_unit.
ele_fiber_ID_analysed	(int, optional): The ID of the analysed element with fibers. If fibers are present in the model and the user wants to save ODB data (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the element with fibers chosen. -1 will ignore the storage of data for fibers. Defaults to -1.
fiber_section	(int, optional): The section number, i.e. the Gauss integratio number. If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
timeSeries_type	(str, optional): Type of timeseries chosen. For more information, see the OpenSeesPy documentation. Defaults to "Linear".
pattern_type	(str, optional): Type of pattern chosen. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
constraints_type	(str, optional): Type of contraints chosen. It detemines how the constraint equations are enforced in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
numberer_type	(str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom. For more information, see the OpenSeesPy documentation. Defaults to "RCM".
system_type	(str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "UmfPack".
analysis_type	(str, optional): Type of analysis chosen. It determines how to construct the Analysis object, which defines what type of analysis is to be performed. For more information, see the OpenSeesPy documentation. Defaults to "Static".
show_plot	(bool, optional): Option to show the 'lateral displacement vs. lateral loading' curve after the analysis. Defaults to True.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Exceptions

NegativeValue	The ID of CtrlNode needs to be a positive integer.
NegativeValue	The ID of timeSeries_ID needs to be a positive integer.
NegativeValue	The ID of pattern_ID needs to be a positive integer.
NegativeValue	The ID of ele_fiber_ID_analysed needs to be a positive integer if is different from -1.

Definition at line 228 of file AnalysisAndPostProcessing.py.

```
00230         show_plot = True, block = False):
00231         """
00232         Method to perform a pushover analysis (displacement-control). If this method is called, the LoadingProtocol and LateralForce methods should be avoided.
00233
00234         @param CtrlNode (int): The node that will be used to impose the displacement Dmax of the pushover analysis.
00235             If the show_plot option is True, the curve displayed follows this node.
00236         @param Dmax (float): The imposed displacement.
00237         @param Dincr (float): The incremental displacement to reach Dmax. To converge, it should be small enough (1000 times smaller of Dmax).
00238         @param timeSeries_ID (int): ID of the timeseries.
00239         @param pattern_ID (int): ID of the pattern.
00240         @param Fx (float, optional): The force imposed at the control node CtrlNode. It is used for convergence reasons and it can be arbitrarly small.
00241             Defaults to 1*kN_unit.
00242         @param ele_fiber_ID_analysed (int, optional): The ID of the analysed element with fibers. If fibers are present in the model and the user wants to save ODB data
00243             (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the element with fibers chosen.
00244             -1 will ignore the storage of data for fibers. Defaults to -1.
00245         @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
00246             If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
00247         @param timeSeries_type (str, optional): Type of timeseries chosen.
00248             For more information, see the OpenSeesPy documentation. Defaults to "Linear".
00249         @param pattern_type (str, optional): Type of pattern chosen.
00250             For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00251         @param constraints_type (str, optional): Type of contraints chosen. It detemines how the constraint equations are enforced in the analysis.
00252             For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00253         @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom.
00254             For more information, see the OpenSeesPy documentation. Defaults to "RCM".
00255         @param system_type (str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis.
00256             For more information, see the OpenSeesPy documentation. Defaults to "UmfPack".
00257         @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct the Analysis object, which defines what type of analysis is to be performed.
00258             For more information, see the OpenSeesPy documentation. Defaults to "Static".
00259         @param show_plot (bool, optional): Option to show the 'lateral displacement vs. lateral loading' curve after the analysis. Defaults to True.
00260         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
00261
00262         @exception NegativeValue: The ID of CtrlNode needs to be a positive integer.
00263         @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
00264         @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
00265         @exception NegativeValue: The ID of ele_fiber_ID_analysed needs to be a positive integer if is different from -1.
00266         """
00267         if CtrlNode < 1: raise NegativeValue()
00268         if timeSeries_ID < 1: raise NegativeValue()
00269         if pattern_ID < 1: raise NegativeValue()
00270         if ele_fiber_ID_analysed != -1 and ele_fiber_ID_analysed < 1: raise NegativeValue()
00271
00272         # for mass defined: opsplt.createODB(self.name_ODB, "Pushover", Nmodes = nEigen);
00273         opsplt.createODB(self.name_ODB, "Pushover");
00274         if ele_fiber_ID_analysed != -1: opsplt.saveFiberData2D(self.name_ODB, "Pushover", ele_fiber_ID_analysed, fiber_section)
00275
00276         # Create load pattern
00277         timeSeries(timeSeries_type, timeSeries_ID)
00278         pattern(pattern_type, timeSeries_ID, pattern_ID)
00279         load(CtrlNode, Fx, 0.0, 0.0)    # load(IDNode, Fx, Fy, Mz)
00280         Nsteps = int(abs(Dmax/Dincr))   # number of pushover analysis steps
00281
00282         # Set up analysis options
00283         constraints(constraints_type)   # how it handles boundary conditions
00284         numberer(numberer_type)         # renumber dof's to minimize band-width (optimization)
00285         system(system_type)             # how to store and solve the system of equations in the analysis
```

```
                                        # For static model, BandGeneral, for transient and/or big model, UmfPack
        integrator("LoadControl", 1)    # placeholder
        algorithm("Newton")             # placeholder
        analysis(analysis_type)         # define type of analysis: static for pushover

        # Analysis
        dataPO = np.zeros((Nsteps+1,2))
        next_step = 0
        print("")
        print("Pushover analysis starts")
        for iteration in range(Nsteps):
            next_step = ProgressingPercentage(Nsteps, iteration, next_step)
            convergence = self.__LatDispCtrlLoop(CtrlNode, Dincr, iteration,
                self.algo, self.test_type, self.tol, self.test_opt, self.max_iter, self.allow_smaller_step)
            if convergence != 0: break
            dataPO[iteration+1,0] = nodeDisp(CtrlNode, 1)/mm_unit
            dataPO[iteration+1,1] = getLoadFactor(pattern_ID)*Fx/kN_unit

        if show_plot:
            plt.plot(dataPO[:,0], dataPO[:,1])
            plt.xlabel('Horizontal Displacement [mm]')
            plt.ylabel('Horizontal Load [kN]')
            plt.title('Pushover curve')
            if block:
                plt.show()

        print("")
        print("Pushover complete")
        self.load_case = "Pushover"

        wipe()
```

7.1.4 Member Data Documentation

7.1.4.1 algo

`algo`

Definition at line 51 of file AnalysisAndPostProcessing.py.

7.1.4.2 allow_smaller_step

`allow_smaller_step`

Definition at line 56 of file AnalysisAndPostProcessing.py.

7.1.4.3 data_dir

`data_dir`

Definition at line 49 of file AnalysisAndPostProcessing.py.

7.1.4.4 load_case

```
load_case
```

Definition at line 57 of file AnalysisAndPostProcessing.py.

7.1.4.5 max_iter

```
max_iter
```

Definition at line 55 of file AnalysisAndPostProcessing.py.

7.1.4.6 name_ODB

```
name_ODB
```

Definition at line 50 of file AnalysisAndPostProcessing.py.

7.1.4.7 test_opt

```
test_opt
```

Definition at line 54 of file AnalysisAndPostProcessing.py.

7.1.4.8 test_type

```
test_type
```

Definition at line 52 of file AnalysisAndPostProcessing.py.

7.1.4.9 tol

```
tol
```

Definition at line 53 of file AnalysisAndPostProcessing.py.

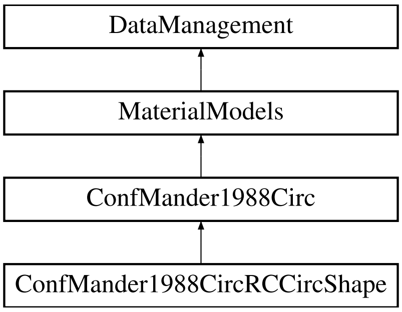
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/AnalysisAndPostProcessing.py

7.2 ConfMander1988Circ Class Reference

Class that stores funcions and material properties of a RC circular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

Inheritance diagram for ConfMander1988Circ:



Public Member Functions

- def __init__ (self, int ID, bc, Ac, fc, Ec, nr_bars, D_bars, s, D_hoops, rho_s_vol, fs, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1)

  Constructor of the class.
- def CheckApplicability (self)

  Implementation of the homonym abstract method.
- def Compute_ec (self)

  Method that computes the compressive concrete yield strain.
- def Compute_ecc (self)

  Method that computes the compressive confined concrete yield strain.
- def Compute_eccu (self)

  Method that computes the compressive confined concrete failure strain.
- def Compute_ecp (self)

  Method that computes the compressive concrete spalling strain.
- def Compute_ecu (self)

  Method that computes the compressive concrete failure strain.
- def Compute_et (self)

  Method that computes the tensile concrete yield strain.
- def Compute_fct (self)

  Method that computes the tensile concrete yield stress.
- def Concrete01 (self)

  Generate the material model Concrete01 for rectangular section confined concrete (Mander 1988).
- def Concrete04 (self)

  Generate the material model Concrete04 for circular section confined concrete (Mander 1988).
- def ReInit (self, ec=1, ecp=1, fct=-1, et=-1)

  Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False, concrete04=True)

  Implementation of the homonym abstract method.
- def UpdateStoredData (self)

  Implementation of the homonym abstract method.

Public Attributes

- Ac
- Acc
- Ae
- bc
- beta
- D_bars
- D_hoops
- data
- Ec
- ec
- ecc
- eccu
- ecp
- ecu
- esu
- et
- fc
- fcc
- fct
- fl
- fl_prime
- fs
- ID
- Initialized
- K_combo
- ke
- nr_bars
- rho_cc
- rho_s_vol
- s
- section_name_tag

7.2.1 Detailed Description

Class that stores funcions and material properties of a RC circular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

For more information about the empirical model for the computation of the parameters, see Mander et Al. 1988, Karthik and Mander 2011 and SIA 262:2012.

Parameters

MaterialModels	Parent abstract class.

Definition at line 1994 of file MaterialModels.py.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 __init__()

```
def __init__ (
              self,
             int ID,
              bc,
              Ac,
              fc,
              Ec,
              nr_bars,
              D_bars,
              s,
              D_hoops,
              rho_s_vol,
              fs,
              ec = 1,
              ecp = 1,
              fct = -1,
              et = -1,
              esu = -1,
              beta = 0.1 )
```

Constructor of the class.

Parameters

ID	(int): Unique material model ID.
bc	(float): Width of the confined core (from the centerline of the hoops, according to Mander et Al. 1988).
Ac	(float): Area of the confined core (according to Mander et Al. 1988).
fc	(float): Compressive concrete yield strength (needs to be negative).
Ec	(float): Young modulus.
nr_bars	(float): Number of reinforcement (allow float for computing the equivalent nr_bars with different reinforcement areas).
D_bars	(float): Diameter of the vertical reinforcing bars.
s	(float): Vertical spacing between hoops.
D_hoops	(float): Diameter of hoops.
rho_s_vol	(float): Compute the ratio of the volume of transverse confining steel to the volume of confined concrete core.
fs	(float): Yield stress for the hoops.
ec	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
ecp	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
fct	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
et	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
esu	(float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
beta	(float, optional): Loating point value defining the exponential curve parameter to define the residual stress. Defaults to 0.1 (according to OpenSeesPy documentation)

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	bc needs to be positive.
NegativeValue	Ac needs to be positive.
PositiveValue	fc needs to be negative.
NegativeValue	Ec needs to be positive.
NegativeValue	nr_bars needs to be positive.
NegativeValue	D_bars needs to be positive.
NegativeValue	s needs to be positive.
NegativeValue	D_hoops needs to be positive.
NegativeValue	rho_s_vol needs to be positive.
NegativeValue	fs needs to be positive.
PositiveValue	ec needs to be negative if different from 1.
PositiveValue	ecp needs to be negative if different from 1.
NegativeValue	fct needs to be positive if different from -1.
NegativeValue	et needs to be positive if different from -1.
NegativeValue	esu needs to be positive if different from -1.

Reimplemented in ConfMander1988CircRCCircShape.

Definition at line 2002 of file MaterialModels.py.

```
02003         ec = 1, ecp = 1, fct = -1, et = -1, esu = -1, beta = 0.1):
02004         """
02005         Constructor of the class.
02006 
02007         @param ID (int): Unique material model ID.
02008         @param bc (float): Width of the confined core (from the centerline of the hoops, according to Mander et Al. 1988).
02009         @param Ac (float): Area of the confined core (according to Mander et Al. 1988).
02010         @param fc (float): Compressive concrete yield strength (needs to be negative).
02011         @param Ec (float): Young modulus.
02012         @param nr_bars (float): Number of reinforcement (allow float for computing the equivalent nr_bars with different reinforcement areas).
02013         @param D_bars (float): Diameter of the vertical reinforcing bars.
02014         @param s (float): Vertical spacing between hoops.
02015         @param D_hoops (float): Diameter of hoops.
02016         @param rho_s_vol (float): Compute the ratio of the volume of transverse confining steel to the volume of confined concrete core.
02017         @param fs (float): Yield stress for the hoops.
02018         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
02019         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
02020         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
02021         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
02022         @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
02023         @param beta (float, optional): Loating point value defining the exponential curve parameter to define the residual stress.
02024             Defaults to 0.1 (according to OpenSeesPy documentation)
02025 
02026         @exception NegativeValue: ID needs to be a positive integer.
02027         @exception NegativeValue: bc needs to be positive.
02028         @exception NegativeValue: Ac needs to be positive.
02029         @exception PositiveValue: fc needs to be negative.
02030         @exception NegativeValue: Ec needs to be positive.
02031         @exception NegativeValue: nr_bars needs to be positive.
02032         @exception NegativeValue: D_bars needs to be positive.
02033         @exception NegativeValue: s needs to be positive.
02034         @exception NegativeValue: D_hoops needs to be positive.
02035         @exception NegativeValue: rho_s_vol needs to be positive.
02036         @exception NegativeValue: fs needs to be positive.
02037         @exception PositiveValue: ec needs to be negative if different from 1.
02038         @exception PositiveValue: ecp needs to be negative if different from 1.
02039         @exception NegativeValue: fct needs to be positive if different from -1.
02040         @exception NegativeValue: et needs to be positive if different from -1.
02041         @exception NegativeValue: esu needs to be positive if different from -1.
```

```
        """
        # Check
        if ID < 0: raise NegativeValue()
        if bc < 0: raise NegativeValue()
        if Ac < 0: raise NegativeValue()
        if fc > 0: raise PositiveValue()
        if Ec < 0: raise NegativeValue()
        if nr_bars < 0: raise NegativeValue()
        if D_bars < 0: raise NegativeValue()
        if s < 0: raise NegativeValue()
        if D_hoops < 0: raise NegativeValue()
        if rho_s_vol < 0: raise NegativeValue()
        if fs < 0: raise NegativeValue()
        if ec != 1 and ec > 0: raise PositiveValue()
        if ecp != 1 and ecp > 0: raise PositiveValue()
        if fct != -1 and fct < 0: raise NegativeValue()
        if et != -1 and et < 0: raise NegativeValue()
        if esu != -1 and esu < 0: raise NegativeValue()

        # Arguments
        self.ID = ID
        self.bc = bc
        self.Ac = Ac
        self.fc = fc
        self.Ec = Ec
        self.nr_bars = nr_bars
        self.D_bars = D_bars
        self.s = s
        self.D_hoops = D_hoops
        self.rho_s_vol = rho_s_vol
        self.fs = fs
        self.esu = 0.05 if esu == -1 else esu
        self.beta = beta

        # Initialized the parameters that are dependent from others
        self.section_name_tag = "None"
        self.Initialized = False
        self.ReInit(ec, ecp, fct, et)

```

7.2.3 Member Function Documentation

7.2.3.1 CheckApplicability()

```
def CheckApplicability (
              self )
```

Implementation of the homonym abstract method.

See parent class MaterialModels for detailed information.

Reimplemented from MaterialModels.

Definition at line 2182 of file MaterialModels.py.

```
02182     def CheckApplicability(self):
02183         """
02184         Implementation of the homonym abstract method.
02185         See parent class MaterialModels for detailed information.
02186         """
02187         Check = True
02188         if self.fc < -110*MPa_unit: # Deierlein 1999
02189             Check = False
02190             print("With High Strength concrete (< -110 MPa), a better material model should be used (see Abdesselam et Al. 2019")
02191         if not Check:
02192             print("The validity of the equations is not fullfilled.")
02193             print("!!!!!!! WARNING !!!!!!! Check material model of Confined Mander 1988, ID=", self.ID)
02194             print("")
02195
02196
```

7.2.3.2 Compute_ec()

```
def Compute_ec (
            self )
```

Method that computes the compressive concrete yield strain.

For more information, see Karthik and Mander 2011.

Returns

float: Strain

Definition at line 2197 of file MaterialModels.py.

```
    def Compute_ec(self):
        """
        Method that computes the compressive concrete yield strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        # return -0.002 # Alternative: Mander et Al. 1988
        return -0.0015 + self.fc/MPa_unit/70000 # Karthik Mander 2011


```

7.2.3.3 Compute_ecc()

```
def Compute_ecc (
            self )
```

Method that computes the compressive confined concrete yield strain.

For more information, see Karthik and Mander 2011.

Returns

float: Strain

Definition at line 2249 of file MaterialModels.py.

```
    def Compute_ecc(self):
        """
        Method that computes the compressive confined concrete yield strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        return (1.0 + 5.0 * (self.K_combo-1.0)) * self.ec # Karthik Mander 2011


```

7.2.3.4 Compute_eccu()

```
def Compute_eccu (
              self )
```

Method that computes the compressive confined concrete failure strain.

For more information, see Karthik and Mander 2011.

Returns

float: Strain

Definition at line 2259 of file MaterialModels.py.

```
    def Compute_eccu(self):
        """
        Method that computes the compressive confined concrete failure strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        # return -0.004 + (1.4*(self.rho_s_x+self.rho_s_y)*self.esu*self.fs) / self.fcc # Alternative:
      Prof. Katrin Beyer
        return 5*self.ecc # Karthik Mander 2011


```

7.2.3.5 Compute_ecp()

```
def Compute_ecp (
              self )
```

Method that computes the compressive concrete spalling strain.

For more information, see Mander et Al. 1988.

Returns

float: Strain

Definition at line 2208 of file MaterialModels.py.

```
    def Compute_ecp(self):
        """
        Method that computes the compressive concrete spalling strain.
        For more information, see Mander et Al. 1988.

        @returns float: Strain
        """
        return 2.0*self.ec


```

7.2.3.6 Compute_ecu()

```
def Compute_ecu (
            self )
```

Method that computes the compressive concrete failure strain.

For more information, see Karthik and Mander 2011.

Returns

float: Strain

Definition at line 2238 of file MaterialModels.py.

```
    def Compute_ecu(self):
        """
        Method that computes the compressive concrete failure strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        # return -0.004 # Alternative: Mander et Al. 1988
        return -0.012 - 0.0001 * self.fc/MPa_unit # Karthik Mander 2011


```

7.2.3.7 Compute_et()

```
def Compute_et (
            self )
```

Method that computes the tensile concrete yield strain.

For more information, see Mander et Al. 1988 (eq 45).

Returns

float: Strain.

Definition at line 2228 of file MaterialModels.py.

```
    def Compute_et(self):
        """
        Method that computes the tensile concrete yield strain.
        For more information, see Mander et Al. 1988 (eq 45).

        @returns float: Strain.
        """
        return self.fct/self.Ec


```

7.2.3.8 Compute_fct()

```
def Compute_fct (
              self )
```

Method that computes the tensile concrete yield stress.

For more information, see SIA 262:2012. Assume that the confinement do not play an essential role in tension.

Returns

float: Stress.

Definition at line 2218 of file MaterialModels.py.

```
    def Compute_fct(self):
        """
        Method that computes the tensile concrete yield stress.
        For more information, see SIA 262:2012. Assume that the confinement do not play an essential
     role in tension.

        @returns float: Stress.
        """
        return 0.30 * math.pow(-self.fc/MPa_unit, 2/3) * MPa_unit


```

7.2.3.9 Concrete01()

```
def Concrete01 (
              self )
```

Generate the material model Concrete01 for rectangular section confined concrete (Mander 1988).

See _Concrete01 function for more information. Use this method or Concrete04, not both (only one material model for ID).

Definition at line 2270 of file MaterialModels.py.

```
    def Concrete01(self):
        """
        Generate the material model Concrete01 for rectangular section confined concrete (Mander
     1988).
        See _Concrete01 function for more information. Use this method or Concrete04, not both (only
     one material model for ID).
        """
        _Concrete01(self.ID, self.ecc, self.fcc, self.eccu)
        self.Initialized = True
        self.UpdateStoredData()


```

7.2.3.10 Concrete04()

```
def Concrete04 (
               self )
```

Generate the material model Concrete04 for circular section confined concrete (Mander 1988).

See _Concrete04 function for more information. Use this method or Concrete01, not both (only one material model for ID).

Definition at line 2280 of file MaterialModels.py.

```
02280     def Concrete04(self):
02281         """
02282         Generate the material model Concrete04 for circular section confined concrete (Mander 1988).
02283         See _Concrete04 function for more information. Use this method or Concrete01, not both (only
       one material model for ID).
02284         """
02285         _Concrete04(self.ID, self.fcc, self.ecc, self.eccu, self.Ec, self.fct, self.et, self.beta)
02286         self.Initialized = True
02287         self.UpdateStoredData()
02288 
02289 
```

7.2.3.11 ReInit()

```
def ReInit (
               self,
               ec = 1,
               ecp = 1,
               fct = -1,
               et = -1 )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

ec	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
ecp	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
fct	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
et	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.

Definition at line 2081 of file MaterialModels.py.

```
02081     def ReInit(self, ec = 1, ecp = 1, fct = -1, et = -1):
02082         """
02083         Implementation of the homonym abstract method.
02084         See parent class DataManagement for detailed information.
02085 
02086         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
       according to Karthik and Mander 2011.
02087         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
       to Mander 1988.
02088         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
       according to SIA 262:2012.
02089         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
       according to SIA 262:2012.
02090         """
02091         # Check applicability
```

```
        self.CheckApplicability()

        # Arguments
        self.ec = self.Compute_ec() if ec == 1 else ec
        self.ecp = self.Compute_ecp() if ecp == 1 else ecp
        self.fct = self.Compute_fct() if fct == -1 else fct
        self.et = self.Compute_et() if et == -1 else et

        # Members
        s_prime = self.s - self.D_hoops
        self.ecu = self.Compute_ecu()
        self.Ae = math.pi/4 * (self.bc - s_prime/2)**2
        self.rho_cc = self.nr_bars*self.D_bars**2/4.0*math.pi / self.Ac
        self.Acc = self.Ac*(1.0-self.rho_cc)
        self.ke = self.Ae/self.Acc
        self.fl = -self.rho_s_vol * self.fs / 2
        self.fl_prime = self.fl * self.ke
        self.K_combo = -1.254 + 2.254 * math.sqrt(1.0+7.94*self.fl_prime/self.fc) - 2.0*self.fl_prime/self.fc
        self.fcc = self.fc * self.K_combo
        self.ecc = self.Compute_ecc()
        self.eccu = self.Compute_eccu()
        if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredData()


```

7.2.3.12 ShowInfo()

```
def ShowInfo (
        self,
        plot = False,
        block = False,
        concrete04 = True )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the material model. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
concrete04	(bool, optional): Option to show in the plot the concrete04 or concrete01 if False. Defaults to True.

Definition at line 2151 of file MaterialModels.py.

```
02151     def ShowInfo(self, plot = False, block = False, concrete04 = True):
02152         """
02153         Implementation of the homonym abstract method.
02154         See parent class DataManagement for detailed information.
02155 
02156         @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
02157         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
02158         @param concrete04 (bool, optional): Option to show in the plot the concrete04 or concrete01 if False. Defaults to True.
02159         """
02160         print("")
02161         print("Requested info for Confined Mander 1988 (circular) material model Parameters, ID = {}".format(self.ID))
02162         print("Section associated: {} ".format(self.section_name_tag))
02163         print('Concrete strength fc = {} MPa'.format(self.fc/MPa_unit))
02164         print('Concrete strength confined fcc = {} MPa'.format(self.fcc/MPa_unit))
```

```
        print('Strain at maximal strength ec = {}'.format(self.ec))
        print('Strain at maximal strength confined ecc = {}'.format(self.ecc))
        print('Maximal strain ecu = {}'.format(self.ecu))
        print('Maximal strain confined eccu = {}'.format(self.eccu))
        print("")

        if plot:
            fig, ax = plt.subplots()
            if concrete04:
                PlotConcrete04(self.fcc, self.Ec, self.ecc, self.eccu, "C", ax, self.ID)
            else:
                PlotConcrete01(self.fcc, self.ecc, 0.0, self.eccu, ax, self.ID)

            if block:
                plt.show()


```

7.2.3.13 UpdateStoredData()

```
def UpdateStoredData (
              self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 2120 of file MaterialModels.py.

```
02120     def UpdateStoredData(self):
02121         """
02122         Implementation of the homonym abstract method.
02123         See parent class DataManagement for detailed information.
02124         """
02125         self.data = [["INFO_TYPE", "ConfMander1988Circ"], # Tag for differentiating different data
02126             ["ID", self.ID],
02127             ["section_name_tag", self.section_name_tag],
02128             ["bc", self.bc],
02129             ["Ac", self.Ac],
02130             ["fc", self.fc],
02131             ["Ec", self.Ec],
02132             ["ec", self.ec],
02133             ["ecp", self.ecp],
02134             ["ecu", self.ecu],
02135             ["fct", self.fct],
02136             ["et", self.et],
02137             ["fcc", self.fcc],
02138             ["ecc", self.ecc],
02139             ["eccu", self.eccu],
02140             ["beta", self.beta],
02141             ["nr_bars", self.nr_bars],
02142             ["D_bars", self.D_bars],
02143             ["s", self.s],
02144             ["D_hoops", self.D_hoops],
02145             ["rho_s_vol", self.rho_s_vol],
02146             ["fs", self.fs],
02147             ["esu", self.esu],
02148             ["Initialized", self.Initialized]]
02149
02150
```

7.2.4 Member Data Documentation

7.2.4.1 Ac

```
Ac
```

Definition at line 2064 of file MaterialModels.py.

7.2.4.2 Acc

```
Acc
```

Definition at line 2105 of file MaterialModels.py.

7.2.4.3 Ae

```
Ae
```

Definition at line 2103 of file MaterialModels.py.

7.2.4.4 bc

```
bc
```

Definition at line 2063 of file MaterialModels.py.

7.2.4.5 beta

```
beta
```

Definition at line 2074 of file MaterialModels.py.

7.2.4.6 D_bars

```
D_bars
```

Definition at line 2068 of file MaterialModels.py.

7.2.4.7 D_hoops

```
D_hoops
```

Definition at line 2070 of file MaterialModels.py.

7.2.4.8 data

```
data
```

Definition at line 2125 of file MaterialModels.py.

7.2.4.9 Ec

```
Ec
```

Definition at line 2066 of file MaterialModels.py.

7.2.4.10 ec

```
ec
```

Definition at line 2095 of file MaterialModels.py.

7.2.4.11 ecc

```
ecc
```

Definition at line 2111 of file MaterialModels.py.

7.2.4.12 eccu

```
eccu
```

Definition at line 2112 of file MaterialModels.py.

7.2.4.13 ecp

```
ecp
```

Definition at line 2096 of file MaterialModels.py.

7.2.4.14 ecu

`ecu`

Definition at line 2102 of file MaterialModels.py.

7.2.4.15 esu

`esu`

Definition at line 2073 of file MaterialModels.py.

7.2.4.16 et

`et`

Definition at line 2098 of file MaterialModels.py.

7.2.4.17 fc

`fc`

Definition at line 2065 of file MaterialModels.py.

7.2.4.18 fcc

`fcc`

Definition at line 2110 of file MaterialModels.py.

7.2.4.19 fct

`fct`

Definition at line 2097 of file MaterialModels.py.

7.2.4.20 fl

`fl`

Definition at line 2107 of file MaterialModels.py.

7.2.4.21 fl_prime

`fl_prime`

Definition at line 2108 of file MaterialModels.py.

7.2.4.22 fs

`fs`

Definition at line 2072 of file MaterialModels.py.

7.2.4.23 ID

`ID`

Definition at line 2062 of file MaterialModels.py.

7.2.4.24 Initialized

`Initialized`

Definition at line 2078 of file MaterialModels.py.

7.2.4.25 K_combo

`K_combo`

Definition at line 2109 of file MaterialModels.py.

7.2.4.26 ke

`ke`

Definition at line 2106 of file MaterialModels.py.

7.2.4.27 nr_bars

`nr_bars`

Definition at line 2067 of file MaterialModels.py.

7.2.4.28 rho_cc

`rho_cc`

Definition at line 2104 of file MaterialModels.py.

7.2.4.29 rho_s_vol

`rho_s_vol`

Definition at line 2071 of file MaterialModels.py.

7.2.4.30 s

`s`

Definition at line 2069 of file MaterialModels.py.

7.2.4.31 section_name_tag

`section_name_tag`

Definition at line 2077 of file MaterialModels.py.

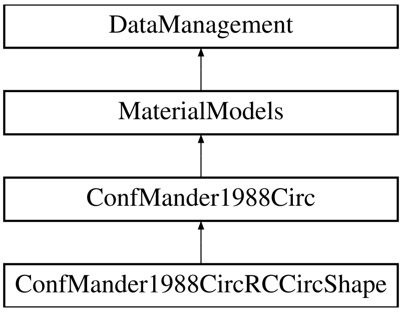
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.3 ConfMander1988CircRCCircShape Class Reference

Class that is the children of ConfMander1988Circ and combine the class RCCircShape (section) to retrieve the information needed.

Inheritance diagram for ConfMander1988CircRCCircShape:



Public Member Functions

- def __init__ (self, int ID, RCCircShape section, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1)

    Constructor of the class.

Public Attributes

- section
- section_name_tag

7.3.1 Detailed Description

Class that is the children of ConfMander1988Circ and combine the class RCCircShape (section) to retrieve the information needed.

Parameters

ConfMander1988Circ	Parent class.

Definition at line 2290 of file MaterialModels.py.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 __init__()

```
def __init__ (
        self,
        int ID,
        RCCircShape section,
        ec = 1,
        ecp = 1,
        fct = -1,
        et = -1,
        esu = -1,
        beta = 0.1 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class RCCircShape. The copy of the section passed is stored in the member variable self.section.

Parameters

ID	(int): Unique material model ID.
section	(RCCircShape): RCCircShape section object.
ec	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
ecp	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
fct	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
et	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
esu	(float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
beta	(float, optional): Loating point value defining the exponential curve parameter to define the residual stress. Defaults to 0.1 (according to OpenSeesPy documentation)

Reimplemented from ConfMander1988Circ.

Definition at line 2296 of file MaterialModels.py.

```
02296     def __init__(self, ID: int, section: RCCircShape, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1):
02297         """
02298         Constructor of the class. It passes the arguments into the parent class to generate the
        combination of the parent class
02299             and the section class RCCircShape.
02300         The copy of the section passed is stored in the member variable self.section.
02301 
02302         @param ID (int): Unique material model ID.
02303         @param section (RCCircShape): RCCircShape section object.
02304         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
        according to Karthik and Mander 2011.
02305         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
        to Mander 1988.
02306         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
        according to SIA 262:2012.
02307         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
        according to SIA 262:2012.
02308         @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed
        according to Mander 1988.
02309         @param beta (float, optional): Loating point value defining the exponential curve parameter to
        define the residual stress.
02310             Defaults to 0.1 (according to OpenSeesPy documentation)
02311         """
02312         self.section = deepcopy(section)
02313         super().__init__(ID, section.bc, section.Ac, section.fc, section.Ec, section.n_bars,
        section.D_bars, section.s, section.D_hoops,
02314             section.rho_s_vol, section.fs, ec=ec, ecp=ecp, fct=fct, et=et, esu=esu, beta=beta)
02315         self.section_name_tag = section.name_tag
```

```
        self.UpdateStoredData()


```

7.3.3 Member Data Documentation

7.3.3.1 section

```
section
```

Definition at line 2312 of file MaterialModels.py.

7.3.3.2 section_name_tag

```
section_name_tag
```

Definition at line 2315 of file MaterialModels.py.

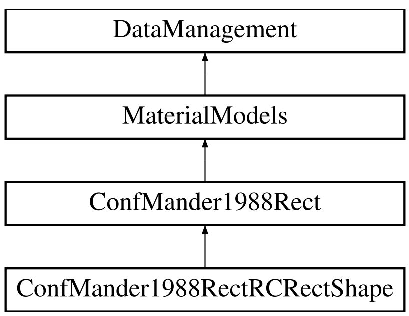
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.4 ConfMander1988Rect Class Reference

Class that stores funcions and material properties of a RC rectangular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

Inheritance diagram for ConfMander1988Rect:



Public Member Functions

- def __init__ (self, int ID, bc, dc, Ac, fc, Ec, nr_bars, D_bars, np.ndarray wx_top, np.ndarray wx_bottom, np.ndarray wy, s, D_hoops, rho_s_x, rho_s_y, fs, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1)

  Constructor of the class.
- def CheckApplicability (self)

  Implementation of the homonym abstract method.
- def Compute_ec (self)

  Method that computes the compressive concrete yield strain.
- def Compute_ecc (self)

  Method that computes the compressive confined concrete yield strain.
- def Compute_eccu (self)

  Method that computes the compressive confined concrete failure strain.
- def Compute_ecp (self)

  Method that computes the compressive concrete spalling strain.
- def Compute_ecu (self)

  Method that computes the compressive concrete failure strain.
- def Compute_et (self)

  Method that computes the tensile concrete yield strain.
- def Compute_fct (self)

  Method that computes the tensile concrete yield stress.
- def ComputeAi (self)

  Method that computes the ineffectual area.
- def ComputeConfinementFactor (self)

  Method that computes the confinement factor using the digitized table from Mander et Al.
- def Concrete01 (self)

  Generate the material model Concrete01 for rectangular section confined concrete (Mander 1988).
- def Concrete04 (self)

  Generate the material model Concrete04 for rectangular section confined concrete (Mander 1988).
- def ReInit (self, ec=1, ecp=1, fct=-1, et=-1)

  Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False, concrete04=True)

  Implementation of the homonym abstract method.
- def UpdateStoredData (self)

  Implementation of the homonym abstract method.

Public Attributes

- Ac
- Acc
- Ae
- Ai
- bc
- beta
- D_bars
- D_hoops
- data
- dc
- Ec
- ec
- ecc

- eccu
- ecp
- ecu
- esu
- et
- fc
- fcc
- fct
- fl_x
- fl_y
- fs
- ID
- Initialized
- K_combo
- ke
- nr_bars
- rho_cc
- rho_s_x
- rho_s_y
- s
- section_name_tag
- wx_bottom
- wx_top
- wy

Static Public Attributes

- list array_fl2 = [None] ∗ len(curve_fl1)
- curve_fl1 = np.arange(0, 0.3+0.02, 0.02)

7.4.1 Detailed Description

Class that stores funcions and material properties of a RC rectangular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

For more information about the empirical model for the computation of the parameters, see Mander et Al. 1988, Karthik and Mander 2011 and SIA 262:2012. The array array_fl2 and curve curve_fl1 are the parameter of the digitized table used to extrapolate the confinement factor; they are used as global throughout the ConfMander1988Rect material model to optimise the program (given the fact that is constant everytime).

Parameters

MaterialModels	Parent abstract class.

Definition at line 1381 of file MaterialModels.py.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 __init__()

```
def __init__ (
              self,
             int ID,
              bc,
              dc,
              Ac,
              fc,
              Ec,
              nr_bars,
              D_bars,
             np.ndarray wx_top,
             np.ndarray wx_bottom,
             np.ndarray wy,
              s,
              D_hoops,
              rho_s_x,
              rho_s_y,
              fs,
              ec = 1,
              ecp = 1,
              fct = -1,
              et = -1,
              esu = -1,
              beta = 0.1 )
```

Constructor of the class.

Parameters

ID	(int): Unique material model ID.
bc	(float): Width of the confined core (from the centerline of the hoops, according to Mander et Al. 1988).
dc	(float): Depth of the confined core (from the centerline of the hoops, according to Mander et Al. 1988).
Ac	(float): Area of the confined core (according to Mander et Al. 1988).
fc	(float): Compressive concrete yield strength (needs to be negative).
Ec	(float): Young modulus.
nr_bars	(float): Number of reinforcement (allow float for computing the equivalent nr_bars with different reinforcement areas).
D_bars	(float): Diameter of the vertical reinforcing bars.
wx_top	(np.ndarray): Vector of 1 dimension that defines the distance between top vertical bars in x direction (NOT CENTERLINE DISTANCES).
wx_bottom	(np.ndarray): Vector of 1 dimension that defines the distance between bottom vertical bars in x direction (NOT CENTERLINE DISTANCES).
wy	(np.ndarray): Vector of 1 dimension that defines the distance between vertical bars in y direction (lateral) (NOT CENTERLINE DISTANCES).
s	(float): Vertical spacing between hoops.
D_hoops	(float): Diameter of hoops.
rho_s_x	(float): Ratio of the transversal area of the hoops to the associated concrete area in the x direction.
rho_s_y	(float): Ratio of the transversal area of the hoops to the associated concrete area in the y direction.
fs	(float): Yield stress for the hoops.

Parameters

ec	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
ecp	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
fct	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
et	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
esu	(float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
beta	(float, optional): Loating point value defining the exponential curve parameter to define the residual stress. Defaults to 0.1 (according to OpenSeesPy documentation)

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	bc needs to be positive.
NegativeValue	dc needs to be positive.
NegativeValue	Ac needs to be positive.
PositiveValue	fc needs to be negative.
NegativeValue	Ec needs to be positive.
NegativeValue	nr_bars needs to be positive.
NegativeValue	D_bars needs to be positive.
NegativeValue	s needs to be positive.
NegativeValue	D_hoops needs to be positive.
NegativeValue	rho_s_x needs to be positive.
NegativeValue	rho_s_y needs to be positive.
NegativeValue	fs needs to be positive.
PositiveValue	ec needs to be negative if different from 1.
PositiveValue	ecp needs to be negative if different from 1.
NegativeValue	fct needs to be positive if different from -1.
NegativeValue	et needs to be positive if different from -1.
NegativeValue	esu needs to be positive if different from -1.

Reimplemented in ConfMander1988RectRCRectShape.

Definition at line 1549 of file MaterialModels.py.

```
        ec = 1, ecp = 1, fct = -1, et = -1, esu = -1, beta = 0.1):
        """
        Constructor of the class.

        @param ID (int): Unique material model ID.
        @param bc (float): Width of the confined core (from the centerline of the hoops, according to Mander et Al. 1988).
        @param dc (float): Depth of the confined core (from the centerline of the hoops, according to Mander et Al. 1988).
        @param Ac (float): Area of the confined core (according to Mander et Al. 1988).
        @param fc (float): Compressive concrete yield strength (needs to be negative).
        @param Ec (float): Young modulus.
        @param nr_bars (float): Number of reinforcement (allow float for computing the equivalent nr_bars with different reinforcement areas).
        @param D_bars (float): Diameter of the vertical reinforcing bars.
        @param wx_top (np.ndarray): Vector of 1 dimension that defines the distance between top vertical bars in x direction (NOT CENTERLINE DISTANCES).
        @param wx_bottom (np.ndarray): Vector of 1 dimension that defines the distance between bottom vertical bars in x direction (NOT CENTERLINE DISTANCES).
        @param wy (np.ndarray): Vector of 1 dimension that defines the distance between vertical bars in y direction (lateral) (NOT CENTERLINE DISTANCES).
```

```
        @param s (float): Vertical spacing between hoops.
        @param D_hoops (float): Diameter of hoops.
        @param rho_s_x (float): Ratio of the transversal area of the hoops to the associated concrete area in the x direction.
        @param rho_s_y (float): Ratio of the transversal area of the hoops to the associated concrete area in the y direction.
        @param fs (float): Yield stress for the hoops.
        @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
        @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
        @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
        @param beta (float, optional): Loating point value defining the exponential curve parameter to define the residual stress.
            Defaults to 0.1 (according to OpenSeesPy documentation)

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: bc needs to be positive.
        @exception NegativeValue: dc needs to be positive.
        @exception NegativeValue: Ac needs to be positive.
        @exception PositiveValue: fc needs to be negative.
        @exception NegativeValue: Ec needs to be positive.
        @exception NegativeValue: nr_bars needs to be positive.
        @exception NegativeValue: D_bars needs to be positive.
        @exception NegativeValue: s needs to be positive.
        @exception NegativeValue: D_hoops needs to be positive.
        @exception NegativeValue: rho_s_x needs to be positive.
        @exception NegativeValue: rho_s_y needs to be positive.
        @exception NegativeValue: fs needs to be positive.
        @exception PositiveValue: ec needs to be negative if different from 1.
        @exception PositiveValue: ecp needs to be negative if different from 1.
        @exception NegativeValue: fct needs to be positive if different from -1.
        @exception NegativeValue: et needs to be positive if different from -1.
        @exception NegativeValue: esu needs to be positive if different from -1.
        """
        # Check
        if ID < 1: raise NegativeValue()
        if bc < 0: raise NegativeValue()
        if dc < 0: raise NegativeValue()
        if Ac < 0: raise NegativeValue()
        if fc > 0: raise PositiveValue()
        if Ec < 0: raise NegativeValue()
        if nr_bars < 0: raise NegativeValue()
        if D_bars < 0: raise NegativeValue()
        if s < 0: raise NegativeValue()
        if D_hoops < 0: raise NegativeValue()
        if rho_s_x < 0: raise NegativeValue()
        if rho_s_y < 0: raise NegativeValue()
        if fs < 0: raise NegativeValue()
        if ec != 1 and ec > 0: raise PositiveValue()
        if ecp != 1 and ecp > 0: raise PositiveValue()
        if fct != -1 and fct < 0: raise NegativeValue()
        if et != -1 and et < 0: raise NegativeValue()
        if esu != -1 and esu < 0: raise NegativeValue()

        # Arguments
        self.ID = ID
        self.bc = bc
        self.dc = dc
        self.Ac = Ac
        self.fc = fc
        self.Ec = Ec
        self.nr_bars = nr_bars
        self.D_bars = D_bars
        self.wx_top = copy(wx_top)
        self.wx_bottom = copy(wx_bottom)
        self.wy = copy(wy)
        self.s = s
        self.D_hoops = D_hoops
        self.rho_s_x = rho_s_x
        self.rho_s_y = rho_s_y
        self.fs = fs
        self.esu = 0.05 if esu == -1 else esu # Mander 1988
        self.beta = beta

        # Initialized the parameters that are dependent from others
        self.section_name_tag = "None"
        self.Initialized = False
        self.ReInit(ec, ecp, fct, et)

```

7.4.3 Member Function Documentation

7.4.3.1 CheckApplicability()

```
def CheckApplicability (
                self )
```

Implementation of the homonym abstract method.

See parent class MaterialModels for detailed information.

Reimplemented from MaterialModels.

Definition at line 1756 of file MaterialModels.py.

```
    def CheckApplicability(self):
        """
        Implementation of the homonym abstract method.
        See parent class MaterialModels for detailed information.
        """
        Check = True
        if self.fc < -110*MPa_unit: # Deierlein 1999
            Check = False
            print("With High Strength concrete (< -110 MPa), a better material model should be used
      (see Abdesselam et Al. 2019")
        if not Check:
            print("The validity of the equations is not fullfilled.")
            print("!!!!!!! WARNING !!!!!!! Check material model of Confined Mander 1988, ID=",
      self.ID)
            print("")


```

7.4.3.2 Compute_ec()

```
def Compute_ec (
                self )
```

Method that computes the compressive concrete yield strain.

For more information, see Karthik and Mander 2011.

Returns

float: Strain

Definition at line 1771 of file MaterialModels.py.

```
    def Compute_ec(self):
        """
        Method that computes the compressive concrete yield strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        # return -0.002 # Alternative: Mander et Al. 1988
        return -0.0015 + self.fc/MPa_unit/70000 # Karthik Mander 2011


```

7.4.3.3 Compute_ecc()

```
def Compute_ecc (
              self )
```

Method that computes the compressive confined concrete yield strain.

For more information, see Karthik and Mander 2011.

Returns

float: Strain

Definition at line 1823 of file MaterialModels.py.

```
    def Compute_ecc(self):
        """
        Method that computes the compressive confined concrete yield strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        return (1.0 + 5.0 * (self.K_combo-1.0)) * self.ec # Karthik Mander 2011


```

7.4.3.4 Compute_eccu()

```
def Compute_eccu (
              self )
```

Method that computes the compressive confined concrete failure strain.

For more information, see Karthik and Mander 2011.

Returns

float: Strain

Definition at line 1833 of file MaterialModels.py.

```
    def Compute_eccu(self):
        """
        Method that computes the compressive confined concrete failure strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        # return -0.004 + (1.4*(self.rho_s_x+self.rho_s_y)*self.esu*self.fs) / self.fcc # Alternative:
  Prof. Katrin Beyer
        return 5*self.ecc # Karthik Mander 2011


```

7.4.3.5 Compute_ecp()

```
def Compute_ecp (
                self )
```

Method that computes the compressive concrete spalling strain.

For more information, see Mander et Al. 1988.

Returns

float: Strain

Definition at line 1782 of file MaterialModels.py.

```
01782     def Compute_ecp(self):
01783         """
01784         Method that computes the compressive concrete spalling strain.
01785         For more information, see Mander et Al. 1988.
01786 
01787         @returns float: Strain
01788         """
01789         return 2.0*self.ec
01790 
01791 
```

7.4.3.6 Compute_ecu()

```
def Compute_ecu (
                self )
```

Method that computes the compressive concrete failure strain.

For more information, see Karthik and Mander 2011.

Returns

float: Strain

Definition at line 1812 of file MaterialModels.py.

```
01812     def Compute_ecu(self):
01813         """
01814         Method that computes the compressive concrete failure strain.
01815         For more information, see Karthik and Mander 2011.
01816 
01817         @returns float: Strain
01818         """
01819         # return -0.004 # Alternative: Mander et Al. 1988
01820         return -0.012 - 0.0001 * self.fc/MPa_unit # Karthik Mander 2011
01821 
01822 
```

7.4.3.7 Compute_et()

```
def Compute_et (
                self )
```

Method that computes the tensile concrete yield strain.

For more information, see Mander et Al. 1988 (eq 45).

Returns

float: Strain.

Definition at line 1802 of file MaterialModels.py.

```
    def Compute_et(self):
        """
        Method that computes the tensile concrete yield strain.
        For more information, see Mander et Al. 1988 (eq 45).

        @returns float: Strain.
        """
        return self.fct/self.Ec


```

7.4.3.8 Compute_fct()

```
def Compute_fct (
                self )
```

Method that computes the tensile concrete yield stress.

For more information, see SIA 262:2012. Assume that the confinement do not play an essential role in tension.

Returns

float: Stress.

Definition at line 1792 of file MaterialModels.py.

```
    def Compute_fct(self):
        """
        Method that computes the tensile concrete yield stress.
        For more information, see SIA 262:2012. Assume that the confinement do not play an essential
 role in tension.

        @returns float: Stress.
        """
        return 0.30 * math.pow(-self.fc/MPa_unit, 2/3) * MPa_unit


```

7.4.3.9 ComputeAi()

```
def ComputeAi (
              self )
```

Method that computes the ineffectual area.

For more information, see Mander et Al. 1988.

Returns

float: Area.

Definition at line 1844 of file MaterialModels.py.

```
01844     def ComputeAi(self):
01845         """
01846         Method that computes the ineffectual area.
01847         For more information, see Mander et Al. 1988.
01848 
01849         @returns float: Area.
01850         """
01851         return ( np.sum(np.multiply(self.wy, self.wy))*2.0 +
01852             np.sum(np.multiply(self.wx_top, self.wx_top)) +
01853             np.sum(np.multiply(self.wx_bottom, self.wx_bottom)) ) / 6.0
01854 
01855 
```

7.4.3.10 ComputeConfinementFactor()

```
def ComputeConfinementFactor (
              self )
```

Method that computes the confinement factor using the digitized table from Mander et Al.

1988 that extrapolates the factor using the lateral confining stress in the two direction.

Exceptions

NoApplicability	The table from Mander accept ratio of fl/fc smaller than 0.3.
NoApplicability	The table from Mander accept ratio of fl/fc smaller than 0.3.
NegativeValue	fl1_ratio needs to be positive.
NegativeValue	fl2_ratio needs to be positive.

Returns

float: Confinement factor.

Definition at line 1856 of file MaterialModels.py.

```
01856     def ComputeConfinementFactor(self):
01857         """
01858         Method that computes the confinement factor using the digitized table from Mander et Al. 1988
      that
01859             extrapolates the factor using the lateral confining stress in the two direction.
01860 
01861         @exception NoApplicability: The table from Mander accept ratio of fl/fc smaller than 0.3.
01862         @exception NoApplicability: The table from Mander accept ratio of fl/fc smaller than 0.3.
```

```
        @exception NegativeValue: fl1_ratio needs to be positive.
        @exception NegativeValue: fl2_ratio needs to be positive.

        @returns float: Confinement factor.
        """
        if self.fl_x == self.fl_y:
            return -1.254 + 2.254 * math.sqrt(1.0+7.94*self.fl_x*self.ke/self.fc) - 2.0*self.fl_x*self.ke/self.fc # in Mander, it has a prime
        else:
            fl2_ratio = max(self.fl_x*self.ke/self.fc, self.fl_y*self.ke/self.fc)
            fl1_ratio = min(self.fl_x*self.ke/self.fc, self.fl_y*self.ke/self.fc)

            if fl1_ratio > 0.3: raise NoApplicability()
            if fl2_ratio > 0.3: raise NoApplicability()
            if fl1_ratio < 0: raise NegativeValue()
            if fl2_ratio < 0: raise NegativeValue()

            # choose one or two curves
            for ii, fl1 in enumerate(curve_fl1):
                if fl1 == fl1_ratio:
                    # one curve
                    # choose curve
                    # curve_fl2 = [curve for ii, curve in enumerate(array_fl2) if index[ii]][0]
                    curve_fl2 = array_fl2[ii]

                    # Take value (interpole)
                    K = [item[0] for item in curve_fl2]
                    fl2 = [item[1] for item in curve_fl2]
                    K_res =  np.interp(fl2_ratio, fl2, K)

                    #TODO: to check fucntion:
                    # fig, ax = plt.subplots()
                    # ax.plot(fl2, K, 'k-')
                    # ax.scatter(fl2_ratio, K_res, color='k')
                    # ax.grid()
                    # plt.show()
                    return K_res

                # two curves
                if fl1 > fl1_ratio:
                    fl1_max = fl1
                    fl1_min = curve_fl1[ii-1]
                    curve_fl2_max = array_fl2[ii]
                    curve_fl2_min = array_fl2[ii-1]

                    # Take the values (interpole)
                    K_max = [item[0] for item in curve_fl2_max]
                    fl2_max = [item[1] for item in curve_fl2_max]
                    K_res_max =  np.interp(fl2_ratio, fl2_max, K_max)

                    K_min = [item[0] for item in curve_fl2_min]
                    fl2_min = [item[1] for item in curve_fl2_min]
                    K_res_min =  np.interp(fl2_ratio, fl2_min, K_min)

                    # interpole with distance from fl1 for fl2
                    # should be logarithmic interpolation but error negligibile
                    K_res = np.interp(fl1_ratio, [fl1_min, fl1_max], [K_res_min, K_res_max])
                    return K_res
```

7.4.3.11 Concrete01()

```
def Concrete01 (
              self )
```

Generate the material model Concrete01 for rectangular section confined concrete (Mander 1988).

See _Concrete01 function for more information. Use this method or Concrete04, not both (only one material model for ID).

Definition at line 1922 of file MaterialModels.py.

```
    def Concrete01(self):
        """
        Generate the material model Concrete01 for rectangular section confined concrete (Mander 1988).
```

```
        See _Concrete01 function for more information. Use this method or Concrete04, not both (only
    one material model for ID).
        """
        _Concrete01(self.ID, self.ecc, self.fcc, self.eccu)
        self.Initialized = True
        self.UpdateStoredData()


```

7.4.3.12 Concrete04()

```
def Concrete04 (
              self )
```

Generate the material model Concrete04 for rectangular section confined concrete (Mander 1988).

See _Concrete04 function for more information. Use this method or Concrete01, not both (only one material model for ID).

Definition at line 1932 of file MaterialModels.py.

```
    def Concrete04(self):
        """
        Generate the material model Concrete04 for rectangular section confined concrete (Mander
    1988).
        See _Concrete04 function for more information. Use this method or Concrete01, not both (only
    one material model for ID).
        """
        _Concrete04(self.ID, self.fcc, self.ecc, self.eccu, self.Ec, self.fct, self.et, self.beta)
        self.Initialized = True
        self.UpdateStoredData()


```

7.4.3.13 ReInit()

```
def ReInit (
              self,
              ec = 1,
              ecp = 1,
              fct = -1,
              et = -1 )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

ec	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
ecp	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
fct	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
et	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.

Definition at line 1642 of file MaterialModels.py.

```
    def ReInit(self, ec = 1, ecp = 1, fct = -1, et = -1):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
        @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
        @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        """
        # Check applicability
        self.CheckApplicability()

        # Arguments
        self.ec = self.Compute_ec() if ec == 1 else ec
        self.ecp = self.Compute_ecp() if ecp == 1 else ecp
        self.fct = self.Compute_fct() if fct == -1 else fct
        self.et = self.Compute_et() if et == -1 else et

        # Members (according to Mander 1988, confined concrete)
        self.ecu = self.Compute_ecu()
        self.Ai = self.ComputeAi()
        self.Ae = (self.Ac - self.Ai) * (1.0 - (self.s-self.D_hoops)/2.0/self.bc)*(1.0 - (self.s-self.D_hoops)/2.0/self.dc)
        self.rho_cc = self.nr_bars*self.D_bars**2/4.0*math.pi / self.Ac
        self.Acc = self.Ac*(1.0-self.rho_cc)
        self.ke = self.Ae/self.Acc
        self.fl_x = -self.rho_s_x * self.fs
        self.fl_y = -self.rho_s_y * self.fs
        self.K_combo = self.ComputeConfinementFactor()
        self.fcc = self.fc * self.K_combo
        self.ecc = self.Compute_ecc()
        self.eccu = self.Compute_eccu()
        if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredData()


```

7.4.3.14 ShowInfo()

```
def ShowInfo (
        self,
        plot = False,
        block = False,
        concrete04 = True )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the material model. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
concrete04	(bool, optional): Option to show in the plot the concrete04 or concrete01 if False. Defaults to True.

Definition at line 1725 of file MaterialModels.py.

```
01725     def ShowInfo(self, plot = False, block = False, concrete04 = True):
01726         """
01727         Implementation of the homonym abstract method.
```

```
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        @param concrete04 (bool, optional): Option to show in the plot the concrete04 or concrete01 if False. Defaults to True.
        """
        print("")
        print("Requested info for Confined Mander 1988 (rectangular) material model Parameters, ID = {}".format(self.ID))
        print("Section associated: {} ".format(self.section_name_tag))
        print('Concrete strength fc = {} MPa'.format(self.fc/MPa_unit))
        print('Concrete strength confined fcc = {} MPa'.format(self.fcc/MPa_unit))
        print('Strain at maximal strength ec = {}'.format(self.ec))
        print('Strain at maximal strength confined ecc = {}'.format(self.ecc))
        print('Maximal strain ecu = {}'.format(self.ecu))
        print('Maximal strain confined eccu = {}'.format(self.eccu))
        print("")

        if plot:
            fig, ax = plt.subplots()
            if concrete04:
                PlotConcrete04(self.fcc, self.Ec, self.ecc, self.eccu, "C", ax, self.ID)
            else:
                PlotConcrete01(self.fcc, self.ecc, 0.0, self.eccu, ax, self.ID)

            if block:
                plt.show()


```

7.4.3.15 UpdateStoredData()

def UpdateStoredData (
    self)

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 1681 of file MaterialModels.py.

```
01681     def UpdateStoredData(self):
01682         """
01683         Implementation of the homonym abstract method.
01684         See parent class DataManagement for detailed information.
01685         """
01686         self.data = [["INFO_TYPE", "ConfMander1988rect"], # Tag for differentiating different data
01687             ["ID", self.ID],
01688             ["section_name_tag", self.section_name_tag],
01689             ["bc", self.bc],
01690             ["dc", self.dc],
01691             ["Ac", self.Ac],
01692             ["fc", self.fc],
01693             ["Ec", self.Ec],
01694             ["ec", self.ec],
01695             ["ecp", self.ecp],
01696             ["ecu", self.ecu],
01697             ["fct", self.fct],
01698             ["et", self.et],
01699             ["fcc", self.fcc],
01700             ["ecc", self.ecc],
01701             ["eccu", self.eccu],
01702             ["beta", self.beta],
01703             ["nr_bars", self.nr_bars],
01704             ["D_bars", self.D_bars],
01705             ["wx_top", self.wx_top],
01706             ["wx_bottom", self.wx_bottom],
01707             ["wy", self.wy],
01708             ["s", self.s],
01709             ["D_hoops", self.D_hoops],
01710             ["rho_s_x", self.rho_s_x],
01711             ["rho_s_y", self.rho_s_y],
01712             ["fs", self.fs],
01713             ["esu", self.esu],
```

```
                ["Ai", self.Ai],
                ["Ae", self.Ae],
                ["rho_cc", self.rho_cc],
                ["Acc", self.Acc],
                ["ke", self.ke],
                ["fl_x", self.fl_x],
                ["fl_y", self.fl_y],
                ["K_combo", self.K_combo],
                ["Initialized", self.Initialized]]
```

7.4.4 Member Data Documentation

7.4.4.1 Ac

```
Ac
```

Definition at line 1621 of file MaterialModels.py.

7.4.4.2 Acc

```
Acc
```

Definition at line 1666 of file MaterialModels.py.

7.4.4.3 Ae

```
Ae
```

Definition at line 1664 of file MaterialModels.py.

7.4.4.4 Ai

```
Ai
```

Definition at line 1663 of file MaterialModels.py.

7.4.4.5 array_fl2

```
list array_fl2 = [None] * len(curve_fl1) [static]
```

Definition at line 1394 of file MaterialModels.py.

7.4.4.6 bc

```
bc
```

Definition at line 1619 of file MaterialModels.py.

7.4.4.7 beta

```
beta
```

Definition at line 1635 of file MaterialModels.py.

7.4.4.8 curve_fl1

```
curve_fl1 = np.arange(0, 0.3+0.02, 0.02) [static]
```

Definition at line 1393 of file MaterialModels.py.

7.4.4.9 D_bars

```
D_bars
```

Definition at line 1625 of file MaterialModels.py.

7.4.4.10 D_hoops

```
D_hoops
```

Definition at line 1630 of file MaterialModels.py.

7.4.4.11 data

```
data
```

Definition at line 1686 of file MaterialModels.py.

7.4.4.12 dc

```
dc
```

Definition at line 1620 of file MaterialModels.py.

7.4.4.13 Ec

```
Ec
```

Definition at line 1623 of file MaterialModels.py.

7.4.4.14 ec

```
ec
```

Definition at line 1656 of file MaterialModels.py.

7.4.4.15 ecc

```
ecc
```

Definition at line 1672 of file MaterialModels.py.

7.4.4.16 eccu

```
eccu
```

Definition at line 1673 of file MaterialModels.py.

7.4.4.17 ecp

```
ecp
```

Definition at line 1657 of file MaterialModels.py.

7.4.4.18 ecu

```
ecu
```

Definition at line 1662 of file MaterialModels.py.

7.4.4.19 esu

```
esu
```

Definition at line 1634 of file MaterialModels.py.

7.4.4.20 et

```
et
```

Definition at line 1659 of file MaterialModels.py.

7.4.4.21 fc

```
fc
```

Definition at line 1622 of file MaterialModels.py.

7.4.4.22 fcc

```
fcc
```

Definition at line 1671 of file MaterialModels.py.

7.4.4.23 fct

```
fct
```

Definition at line 1658 of file MaterialModels.py.

7.4.4.24 fl_x

```
fl_x
```

Definition at line 1668 of file MaterialModels.py.

7.4.4.25 fl_y

```
fl_y
```

Definition at line 1669 of file MaterialModels.py.

7.4.4.26 fs

```
fs
```

Definition at line 1633 of file MaterialModels.py.

7.4.4.27 ID

```
ID
```

Definition at line 1618 of file MaterialModels.py.

7.4.4.28 Initialized

```
Initialized
```

Definition at line 1639 of file MaterialModels.py.

7.4.4.29 K_combo

```
K_combo
```

Definition at line 1670 of file MaterialModels.py.

7.4.4.30 ke

```
ke
```

Definition at line 1667 of file MaterialModels.py.

7.4.4.31 nr_bars

```
nr_bars
```

Definition at line 1624 of file MaterialModels.py.

7.4.4.32 rho_cc

```
rho_cc
```

Definition at line 1665 of file MaterialModels.py.

7.4.4.33 rho_s_x

```
rho_s_x
```

Definition at line 1631 of file MaterialModels.py.

7.4.4.34 rho_s_y

```
rho_s_y
```

Definition at line 1632 of file MaterialModels.py.

7.4.4.35 s

```
s
```

Definition at line 1629 of file MaterialModels.py.

7.4.4.36 section_name_tag

```
section_name_tag
```

Definition at line 1638 of file MaterialModels.py.

7.4.4.37 wx_bottom

```
wx_bottom
```

Definition at line 1627 of file MaterialModels.py.

7.4.4.38 wx_top

```
wx_top
```

Definition at line 1626 of file MaterialModels.py.

7.4.4.39 wy

```
wy
```

Definition at line 1628 of file MaterialModels.py.

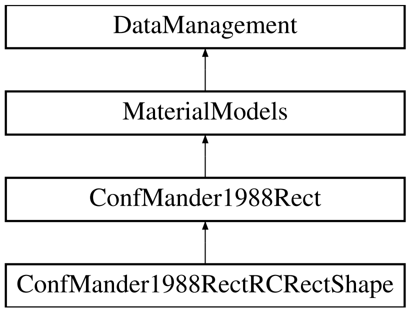
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.5 ConfMander1988RectRCRectShape Class Reference

Class that is the children of ConfMander1988Rect and combine the class RCRectShape (section) to retrieve the information needed.

Inheritance diagram for ConfMander1988RectRCRectShape:



Public Member Functions

- def __init__ (self, int ID, RCRectShape section, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1)

  Constructor of the class.

Public Attributes

- section
- section_name_tag

Additional Inherited Members

7.5.1 Detailed Description

Class that is the children of ConfMander1988Rect and combine the class RCRectShape (section) to retrieve the information needed.

Parameters

ConfMander1988Rect	Parent class.

Definition at line 1942 of file MaterialModels.py.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 __init__()

```
def __init__ (
              self,
             int ID,
             RCRectShape section,
              ec = 1,
              ecp = 1,
              fct = -1,
              et = -1,
              esu = -1,
              beta = 0.1 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class RCRectShape. wx_bottom, wx_top and wy are computed using the private method __Compute_w that and the member variable bars_ranges_position_y and bars_position_x from the section passed. The copy of the section passed is stored in the member variable self.section.

Parameters

ID	(int): Unique material model ID.
section	(RCRectShape): RCRectShape section object.
ec	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
ecp	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
fct	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
et	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
esu	(float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
beta	(float, optional): Loating point value defining the exponential curve parameter to define the residual stress. Defaults to 0.1 (according to OpenSeesPy documentation)

Reimplemented from ConfMander1988Rect.

Definition at line 1948 of file MaterialModels.py.

```
01948     def __init__(self, ID: int, section: RCRectShape, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1):
01949         """
01950         Constructor of the class. It passes the arguments into the parent class to generate the combination of the parent class
01951             and the section class RCRectShape. wx_bottom, wx_top and wy are computed using the private method __Compute_w that
01952             and the member variable bars_ranges_position_y and bars_position_x from the section passed.
01953         The copy of the section passed is stored in the member variable self.section.
01954 
01955         @param ID (int): Unique material model ID.
01956         @param section (RCRectShape): RCRectShape section object.
01957         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
01958         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
01959         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
01960         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
01961         @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
01962         @param beta (float, optional): Loating point value defining the exponential curve parameter to define the residual stress.
01963             Defaults to 0.1 (according to OpenSeesPy documentation)
```

```
01964         """
01965         self.section = deepcopy(section)
01966         ranges = section.bars_ranges_position_y
01967         bars = section.bars_position_x
01968         wy = self.__Compute_w(ranges, section.D_bars)
01969         wx_top = self.__Compute_w(bars[0], section.D_bars)
01970         wx_bottom = self.__Compute_w(bars[-1], section.D_bars)
01971
01972         super().__init__(ID, section.bc, section.dc, section.Ac, section.fc, section.Ec,
      section.nr_bars, section.D_bars,
01973             wx_top, wx_bottom, wy, section.s, section.D_hoops, section.rho_s_x, section.rho_s_y,
      section.fs,
01974             ec=ec, ecp=ecp, fct=fct, et=et, esu=esu, beta=beta)
01975         self.section_name_tag = section.name_tag
01976         self.UpdateStoredData()
01977
```

7.5.3 Member Data Documentation

7.5.3.1 section

`section`

Definition at line 1965 of file MaterialModels.py.

7.5.3.2 section_name_tag

`section_name_tag`

Definition at line 1975 of file MaterialModels.py.

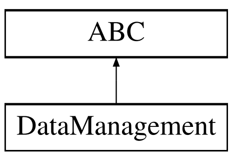
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.6 DataManagement Class Reference

Abstract parent class for data management.

Inheritance diagram for DataManagement:



Public Member Functions

- def ReInit (self)

  Abstract method that computes the value of the parameters with respect of the arguments.
- def SaveData (self, f)

  Function that lists in the command window and saves in a opened file text "f" the data from the "self" class that calls it.
- def ShowInfo (self)

  Abstract method that shows the data stored in the class in the command window.
- def UpdateStoredData (self)

  Abstract method used to define and update the self.data member variable.

7.6.1 Detailed Description

Abstract parent class for data management.

Using the associated MATLAB class
LOAD_CLASS.m
for the postprocessing in MATLAB, allowing for simpler and more reliable data management because the parameters from the OpenSeesPy analysis are imported automatically.

Definition at line 11 of file DataManagement.py.

7.6.2 Member Function Documentation

7.6.2.1 ReInit()

```
def ReInit (
        self )
```

Abstract method that computes the value of the parameters with respect of the arguments.

Use after changing the value of argument inside the class (to update the values accordingly).
This function can be very useful in combination with the function "deepcopy()" from the module "copy".
Be careful that the parameter self.Initialized is also copied, thus it is safer to copy the class before the method that calls the actual OpenSees commands (and initialise the object).

Definition at line 56 of file DataManagement.py.

```
00056     def ReInit(self):
00057         """
00058         Abstract method that computes the value of the parameters with respect of the arguments. \n
00059         Use after changing the value of argument inside the class (to update the values accordingly). \n
00060         This function can be very useful in combination with the function "deepcopy()" from the module "copy". \n
00061         Be careful that the parameter self.Initialized is also copied, thus it is safer to copy the class before the method that calls the actual OpenSees commands (and initialise the object).
00062         """
00063         pass
00064 
```

7.6.2.2 SaveData()

```
def SaveData (
                self,
                f )
```

Function that lists in the command window and saves in a opened file text "f" the data from the "self" class that calls it.

Example: call this function after this line:
with open(FileName, 'w') as f:

Parameters

f	(io.TextIOWrapper): Opened file to write into

Exceptions

WrongDimension	The number of lists in the list self.data needs to be 2

Definition at line 20 of file DataManagement.py.

```
00020     def SaveData(self, f):
00021         """
00022         Function that lists in the command window and saves in a opened file text "f" the data from
       the "self" class that calls it.
00023         Example: call this function after this line: \n
00024         with open(FileName, 'w') as f:
00025 
00026         @param f (io.TextIOWrapper): Opened file to write into
00027 
00028         @exception WrongDimension: The number of lists in the list self.data needs to be 2
00029         """
00030         if len(self.data[0]) != 2: raise WrongDimension()
00031 
00032         delimiter = "##############################" # 30 times #
00033         col_delimiter = "\t" # tab
00034         for data_line in self.data:
00035             f.write('\n')
00036             for col in data_line:
00037                 if type(col) == np.ndarray:
00038                     tmp_str = np.array_str(col, max_line_width = np.inf)
00039                 else:
00040                     tmp_str = str(col)
00041                 f.write(tmp_str)
00042                 f.write(col_delimiter)
00043         f.write('\n')
00044         f.write('NEW INFO SECTION DELIMITER \t')
00045         f.write(delimiter)
00046 
```

7.6.2.3 ShowInfo()

```
def ShowInfo (
                self )
```

Abstract method that shows the data stored in the class in the command window.

In some cases, it's possible to plot some information (for example the curve of the material model).

Definition at line 48 of file DataManagement.py.

```
00048     def ShowInfo(self):
```

```
        """
        Abstract method that shows the data stored in the class in the command window.
        In some cases, it's possible to plot some information (for example the curve of the material
    model).
        """
        pass

```

7.6.2.4 UpdateStoredData()

```
def UpdateStoredData (
                self )
```

Abstract method used to define and update the self.data member variable.

This member variable (self.data) is a list of lists with 2 entries (info_name and info_value) and for each list is stored a different member variable of the class.
Useful to debug the model, export data, copy object.

Definition at line 66 of file DataManagement.py.

```
00066     def UpdateStoredData(self):
00067         """
00068         Abstract method used to define and update the self.data member variable. \n
00069         This member variable (self.data) is a list of lists with 2 entries (info_name and info_value)
00070             and for each list is stored a different member variable of the class. \n
00071         Useful to debug the model, export data, copy object.
00072         """
00073         pass
```

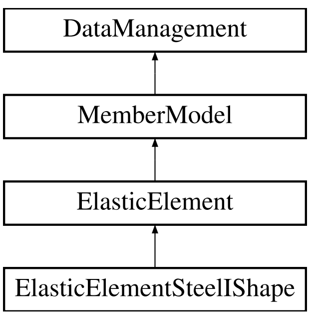
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/DataManagement.py

7.7 ElasticElement Class Reference

Class that handles the storage and manipulation of a elastic element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Inheritance diagram for ElasticElement:



Public Member Functions

- def __init__ (self, int iNode_ID, int jNode_ID, A, E, Iy, int geo_transf_ID, ele_ID=-1)

  Constructor of the class.
- def CreateMember (self)

  Method that initialises the member by calling the OpenSeesPy commands through various functions.
- def Record (self, str name_txt, str data_dir, force_rec=True, def_rec=True, time_rec=True)

  Implementation of the homonym abstract method.
- def RecordNodeDef (self, str name_txt, str data_dir, time_rec=True)

  Implementation of the homonym abstract method.
- def ReInit (self, ele_ID=-1)

  Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False)

  Implementation of the homonym abstract method.
- def UpdateStoredData (self)

  Implementation of the homonym abstract method.

Public Attributes

- A
- data
- E
- element_array
- element_ID
- geo_transf_ID
- Initialized
- iNode_ID
- Iy
- jNode_ID
- section_name_tag

7.7.1 Detailed Description

Class that handles the storage and manipulation of a elastic element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Parameters

MemberModel	Parent abstract class.

Definition at line 496 of file MemberModel.py.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 __init__()

```
def __init__ (
              self,
              int iNode_ID,
              int jNode_ID,
              A,
              E,
              Iy,
              int geo_transf_ID,
              ele_ID = -1 )
```

Constructor of the class.

Parameters

iNode_ID	(int): ID of the first end node.
jNode_ID	(int): ID of the second end node.
A	(float): Area of the member.
E	(float): Young modulus.
Iy	(float): Second moment of inertia (strong axis).
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	ID needs to be a positive integer.
NegativeValue	A needs to be positive.
NegativeValue	E needs to be positive.
NegativeValue	Iy needs to be positive.
NegativeValue	ID needs to be a positive integer.
NegativeValue	ID needs to be a positive integer.

Reimplemented in ElasticElementSteelIShape.

Definition at line 502 of file MemberModel.py.

```
00502     def __init__(self, iNode_ID: int, jNode_ID: int, A, E, Iy, geo_transf_ID: int, ele_ID = -1):
00503         """
00504         Constructor of the class.
00505 
00506         @param iNode_ID (int): ID of the first end node.
00507         @param jNode_ID (int): ID of the second end node.
00508         @param A (float): Area of the member.
00509         @param E (float): Young modulus.
00510         @param Iy (float): Second moment of inertia (strong axis).
00511         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
       documentation).
00512         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
       IDConvention to define it.
00513 
00514         @exception NegativeValue: ID needs to be a positive integer.
00515         @exception NegativeValue: ID needs to be a positive integer.
00516         @exception NegativeValue: A needs to be positive.
00517         @exception NegativeValue: E needs to be positive.
00518         @exception NegativeValue: Iy needs to be positive.
00519         @exception NegativeValue: ID needs to be a positive integer.
00520         @exception NegativeValue: ID needs to be a positive integer.
00521         """
00522         # Check
00523         if iNode_ID < 1: raise NegativeValue()
```

```
        if jNode_ID < 1: raise NegativeValue()
        if A < 0: raise NegativeValue()
        if E < 0: raise NegativeValue()
        if Iy < 0: raise NegativeValue()
        if geo_transf_ID < 1: raise NegativeValue()
        if ele_ID != -1 and ele_ID < 1: raise NegativeValue()

        # Arguments
        self.iNode_ID = iNode_ID
        self.jNode_ID = jNode_ID
        self.A = A
        self.E = E
        self.Iy = Iy
        self.geo_transf_ID = geo_transf_ID

        # Initialized the parameters that are dependent from others
        self.section_name_tag = "None"
        self.Initialized = False
        self.ReInit(ele_ID = -1)

```

7.7.3 Member Function Documentation

7.7.3.1 CreateMember()

```
def CreateMember (
        self )
```

Method that initialises the member by calling the OpenSeesPy commands through various functions.

Definition at line 605 of file MemberModel.py.

```
00605     def CreateMember(self):
00606         """
00607         Method that initialises the member by calling the OpenSeesPy commands through various
    functions.
00608         """
00609         self.element_array = [[self.element_ID, self.iNode_ID, self.jNode_ID]]
00610
00611         # Define element
00612         element("elasticBeamColumn", self.element_ID, self.iNode_ID, self.jNode_ID, self.A, self.E,
    self.Iy, self.geo_transf_ID)
00613
00614         # Update class
00615         self.Initialized = True
00616         self.UpdateStoredData()
00617
00618
```

7.7.3.2 Record()

```
def Record (
        self,
        str name_txt,
        str data_dir,
        force_rec = True,
        def_rec = True,
        time_rec = True )
```

Implementation of the homonym abstract method.

See parent class MemberModel for detailed information.

Reimplemented from MemberModel.

Definition at line 619 of file MemberModel.py.

```
00619     def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
00620         """
00621         Implementation of the homonym abstract method.
00622         See parent class MemberModel for detailed information.
00623         """
00624         super().Record(self.element_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec,
      time_rec=time_rec)
00625
00626
```

7.7.3.3 RecordNodeDef()

```
def RecordNodeDef (
              self,
             str name_txt,
             str data_dir,
              time_rec = True )
```

Implementation of the homonym abstract method.

See parent class MemberModel for detailed information.

Reimplemented from MemberModel.

Definition at line 627 of file MemberModel.py.

```
00627     def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
00628         """
00629         Implementation of the homonym abstract method.
00630         See parent class MemberModel for detailed information.
00631         """
00632         super().RecordNodeDef(self.iNode_ID, self.jNode_ID, name_txt, data_dir, time_rec=time_rec)
00633
00634
```

7.7.3.4 ReInit()

```
def ReInit (
              self,
              ele_ID = -1 )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Definition at line 544 of file MemberModel.py.

```
00544     def ReInit(self, ele_ID = -1):
00545         """
00546         Implementation of the homonym abstract method.
```

```
        See parent class DataManagement for detailed information.

        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
        """
        # Members
        if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + " (modified)"

        # element ID
        self.element_ID = IDConvention(self.iNode_ID, self.jNode_ID) if ele_ID == -1 else ele_ID

        # Data storage for loading/saving
        self.UpdateStoredData()


```

7.7.3.5 ShowInfo()

```
def ShowInfo (
            self,
            plot = False,
            block = False )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the material model. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 579 of file MemberModel.py.

```
00579     def ShowInfo(self, plot = False, block = False):
00580         """
00581         Implementation of the homonym abstract method.
00582         See parent class DataManagement for detailed information.
00583 
00584         @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
00585         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
00586         """
00587         print("")
00588         print("Requested info for ElasticElement member model, ID = {}".format(self.element_ID))
00589         print("Section associated {} ".format(self.section_name_tag))
00590         print("Area A = {} mm2".format(self.A/mm2_unit))
00591         print("Young modulus E = {} GPa".format(self.E/GPa_unit))
00592         print("Moment of inertia Iy = {} mm4".format(self.Iy/mm4_unit))
00593         print("Geometric transformation = {}".format(self.geo_transf_ID))
00594         print("")
00595 
00596         if plot:
00597             if self.Initialized:
00598                 plot_member(self.element_array, "Elastic Element, ID = {}".format(self.element_ID))
00599                 if block:
00600                     plt.show()
00601             else:
00602                 print("The ElasticElement is not initialized (node and elements not created), ID = {}".format(self.element_ID))
00603 
00604 
```

7.7.3.6 UpdateStoredData()

```
def UpdateStoredData (
              self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 562 of file MemberModel.py.

```
    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.data = [["INFO_TYPE", "ElasticElement"], # Tag for differentiating different data
            ["element_ID", self.element_ID],
            ["section_name_tag", self.section_name_tag],
            ["A", self.A],
            ["E", self.E],
            ["Iy", self.Iy],
            ["iNode_ID", self.iNode_ID],
            ["jNode_ID", self.jNode_ID],
            ["tranf_ID", self.geo_transf_ID],
            ["Initialized", self.Initialized]]


```

7.7.4 Member Data Documentation

7.7.4.1 A

```
A
```

Definition at line 534 of file MemberModel.py.

7.7.4.2 data

```
data
```

Definition at line 567 of file MemberModel.py.

7.7.4.3 E

```
E
```

Definition at line 535 of file MemberModel.py.

7.7.4.4 element_array

`element_array`

Definition at line 609 of file MemberModel.py.

7.7.4.5 element_ID

`element_ID`

Definition at line 555 of file MemberModel.py.

7.7.4.6 geo_transf_ID

`geo_transf_ID`

Definition at line 537 of file MemberModel.py.

7.7.4.7 Initialized

`Initialized`

Definition at line 541 of file MemberModel.py.

7.7.4.8 iNode_ID

`iNode_ID`

Definition at line 532 of file MemberModel.py.

7.7.4.9 Iy

`Iy`

Definition at line 536 of file MemberModel.py.

7.7.4.10 jNode_ID

`jNode_ID`

Definition at line 533 of file MemberModel.py.

7.7.4.11 section_name_tag

`section_name_tag`

Definition at line 540 of file MemberModel.py.

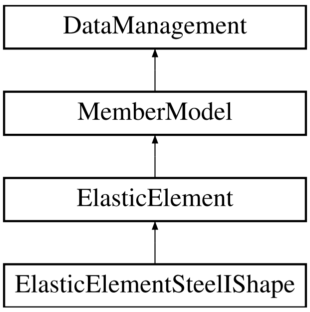
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.8 ElasticElementSteelIShape Class Reference

Class that is the children of ElasticElement and combine the class SteelIShape (section) to retrieve the information needed.

Inheritance diagram for ElasticElementSteelIShape:



Public Member Functions

- def __init__ (self, int iNode_ID, int jNode_ID, SteelIShape section, int geo_transf_ID, ele_ID=-1)
  Constructor of the class.

Public Attributes

- section
- section_name_tag

7.8.1 Detailed Description

Class that is the children of ElasticElement and combine the class SteelIShape (section) to retrieve the information needed.

Parameters

ElasticElement	Parent class.

Definition at line 635 of file MemberModel.py.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 __init__()

```
def __init__ (
              self,
             int iNode_ID,
             int jNode_ID,
             SteelIShape section,
             int geo_transf_ID,
              ele_ID = -1 )
```

Constructor of the class.

Parameters

iNode_ID	(int): ID of the first end node.
jNode_ID	(int): ID of the second end node.
section	(SteelIShape): SteelIShape section object.
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from ElasticElement.

Definition at line 641 of file MemberModel.py.

```
00641     def __init__(self, iNode_ID: int, jNode_ID: int, section: SteelIShape, geo_transf_ID: int, ele_ID
       = -1):
00642         """
00643         Constructor of the class.
00644
00645         @param iNode_ID (int): ID of the first end node.
00646         @param jNode_ID (int): ID of the second end node.
00647         @param section (SteelIShape): SteelIShape section object.
00648         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
       documentation).
00649         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
       IDConvention to define it.
00650         """
00651         self.section = deepcopy(section)
00652         super().__init__(iNode_ID, jNode_ID, section.A, section.E, section.Iy, geo_transf_ID, ele_ID)
00653         self.section_name_tag = section.name_tag
00654         self.UpdateStoredData()
00655         # Check length
00656         self._CheckL()
00657
00658
```

7.8.3 Member Data Documentation

7.8.3.1 section

```
section
```

Definition at line 651 of file MemberModel.py.

7.8.3.2 section_name_tag

```
section_name_tag
```

Definition at line 653 of file MemberModel.py.

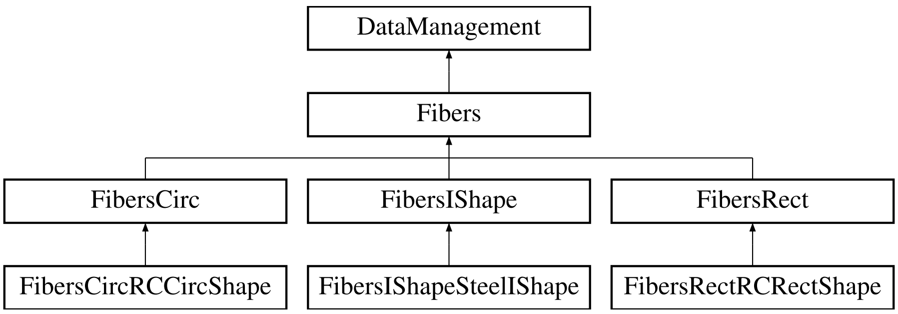
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.9 Fibers Class Reference

Parent abstract class for the storage and manipulation of a fiber's information (mechanical and geometrical parameters, etc) and initialisation in the model.

Inheritance diagram for Fibers:



7.9.1 Detailed Description

Parent abstract class for the storage and manipulation of a fiber's information (mechanical and geometrical parameters, etc) and initialisation in the model.

Parameters

DataManagement	Parent abstract class.

Definition at line 18 of file Fibers.py.

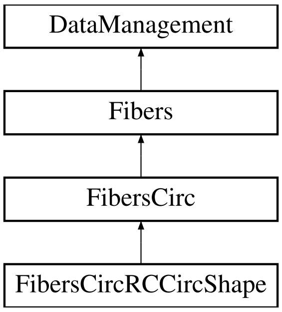
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers.py

7.10 FibersCirc Class Reference

Class that stores funcions, material properties, geometric and mechanical parameters for a circular RC fiber section.

Inheritance diagram for FibersCirc:



Public Member Functions

- def __init__ (self, int ID, b, e, D_bars, Ay, n_bars, D_hoops, int unconf_mat_ID, int conf_mat_ID, int bars_mat_ID, list discr_core, list discr_cover, alpha_i=0.0, GJ=0.0)

    Constructor of the class.
- def CreateFibers (self)

    Method that initialise the fiber by calling the OpenSeesPy commands.
- def ReInit (self)

    Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False)

    Implementation of the homonym abstract method.
- def UpdateStoredData (self)

    Implementation of the homonym abstract method.

Public Attributes

- alpha_i
- Ay
- b
- bars_mat_ID
- conf_mat_ID
- D_bars
- D_hoops
- data
- discr_core
- discr_cover
- e
- fib_sec
- GJ
- ID
- Initialized
- n_bars
- r_bars
- r_core
- section_name_tag
- unconf_mat_ID

7.10.1 Detailed Description

Class that stores funcions, material properties, geometric and mechanical parameters for a circular RC fiber section.

Coordinates: plotting coordinte (x, y) = fiber section coordinate (z, y) = (-x, y). For more information, see the OpenSeesPy documentation.

Parameters

Fibers	Parent abstract class.

Definition at line 263 of file Fibers.py.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 __init__()

```
def __init__ (
              self,
             int ID,
              b,
              e,
              D_bars,
              Ay,
              n_bars,
              D_hoops,
             int unconf_mat_ID,
             int conf_mat_ID,
             int bars_mat_ID,
             list discr_core,
             list discr_cover,
              alpha_i = 0.0,
              GJ = 0.0 )
```

Constructor of the class.

Parameters

ID	(int): Unique fiber section ID.
b	(float): Width of the section.
e	(float): Concrete cover.
D_bars	(float): Diameter of vertical reinforcing bars.
Ay	(float): Area of one vertical reinforcing bar.
n_bars	(float): Number of reinforcement (allow float for computing the equivalent n_bars with different reinforcement areas).
D_hoops	(float): Diameter of the hoops.
unconf_mat_ID	(int): ID of material model that will be assigned to the unconfined fibers.
conf_mat_ID	(int): ID of material model that will be assigned to the confined fibers.
bars_mat_ID	(int): ID of material model that will be assigned to the reinforcing bars fibers.

Parameters

discr_core	(list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges), number of subdivisions (fibers) in the radial direction (number of rings) for the confined core.
discr_cover	(list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges), number of subdivisions (fibers) in the radial direction (number of rings) for the unconfined cover.
alpha_i	(float, optional): Angle in deg of the first vertical rebars with respect to the y axis, counterclockwise. Defaults to 0.0.
GJ	(float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	b needs to be positive.
NegativeValue	e needs to be positive.
InconsistentGeometry	e can't be bigger than half of the width b.
NegativeValue	D_bars needs to be positive.
NegativeValue	Ay needs to be positive.
NegativeValue	n_bars needs to be positive.
NegativeValue	D_hoops needs to be positive.
NegativeValue	unconf_mat_ID needs to be a positive integer.
NegativeValue	conf_mat_ID needs to be a positive integer.
NegativeValue	bars_mat_ID needs to be a positive integer.
WrongDimension	discr_core has a length of 2.
WrongDimension	discr_cover has a length of 2.
NegativeValue	GJ needs to be positive.

Reimplemented in FibersCircRCCircShape.

Definition at line 270 of file Fibers.py.

```
        discr_core: list, discr_cover: list, alpha_i = 0.0, GJ = 0.0):
        """
        Constructor of the class.

        @param ID (int): Unique fiber section ID.
        @param b (float): Width of the section.
        @param e (float): Concrete cover.
        @param D_bars (float): Diameter of vertical reinforcing bars.
        @param Ay (float): Area of one vertical reinforcing bar.
        @param n_bars (float): Number of reinforcement (allow float for computing the equivalent n_bars with different reinforcement areas).
        @param D_hoops (float): Diameter of the hoops.
        @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined fibers.
        @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
        @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars fibers.
        @param discr_core (list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges),
            number of subdivisions (fibers) in the radial direction (number of rings) for the confined core.
        @param discr_cover (list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges),
            number of subdivisions (fibers) in the radial direction (number of rings) for the unconfined cover.
        @param alpha_i (float, optional): Angle in deg of the first vertical rebars with respect to the y axis, counterclockwise. Defaults to 0.0.
        @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

```

```
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: b needs to be positive.
        @exception NegativeValue: e needs to be positive.
        @exception InconsistentGeometry: e can't be bigger than half of the width b.
        @exception NegativeValue: D_bars needs to be positive.
        @exception NegativeValue: Ay needs to be positive.
        @exception NegativeValue: n_bars needs to be positive.
        @exception NegativeValue: D_hoops needs to be positive.
        @exception NegativeValue: unconf_mat_ID needs to be a positive integer.
        @exception NegativeValue: conf_mat_ID needs to be a positive integer.
        @exception NegativeValue: bars_mat_ID needs to be a positive integer.
        @exception WrongDimension: discr_core has a length of 2.
        @exception WrongDimension: discr_cover has a length of 2.
        @exception NegativeValue: GJ needs to be positive.
        """
        # Check
        if ID < 1: raise NegativeValue()
        if b < 0: raise NegativeValue()
        if e < 0: raise NegativeValue()
        if e > b/2: raise InconsistentGeometry()
        if D_bars < 0: raise NegativeValue()
        if Ay < 0: raise NegativeValue()
        if n_bars < 0: raise NegativeValue()
        if D_hoops < 0: raise NegativeValue()
        if unconf_mat_ID < 1: raise NegativeValue()
        if conf_mat_ID < 1: raise NegativeValue()
        if bars_mat_ID < 1: raise NegativeValue()
        if len(discr_core) != 2: raise WrongDimension()
        if len(discr_cover) != 2: raise WrongDimension()
        if GJ < 0: raise NegativeValue()

        # Arguments
        self.ID = ID
        self.b = b
        self.e = e
        self.D_bars = D_bars
        self.Ay = Ay
        self.n_bars = n_bars
        self.D_hoops = D_hoops
        self.unconf_mat_ID = unconf_mat_ID
        self.conf_mat_ID = conf_mat_ID
        self.bars_mat_ID = bars_mat_ID
        self.discr_core = copy(discr_core)
        self.discr_cover = copy(discr_cover)
        self.alpha_i = alpha_i
        self.GJ = GJ

        # Initialized the parameters that are dependent from others
        self.section_name_tag = "None"
        self.Initialized = False
        self.ReInit()

```

7.10.3 Member Function Documentation

7.10.3.1 CreateFibers()

```
def CreateFibers (
                self )
```

Method that initialise the fiber by calling the OpenSeesPy commands.

Definition at line 426 of file Fibers.py.

```
    def CreateFibers(self):
        """
        Method that initialise the fiber by calling the OpenSeesPy commands.
        """
        create_fiber_section(self.fib_sec)
        self.Initialized = True
        self.UpdateStoredData()


```

7.10.3.2 ReInit()

```
def ReInit (
        self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 344 of file Fibers.py.

```
00344     def ReInit(self):
00345         """
00346         Implementation of the homonym abstract method.
00347         See parent class DataManagement for detailed information.
00348         """
00349         # Memebers
00350         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + " (modified)"
00351 
00352         # Parameters
00353         self.r_bars = self.b/2 - self.e - self.D_hoops - self.D_bars/2
00354         self.r_core = self.b/2 - self.e - self.D_hoops/2
00355 
00356         # Create the concrete core fibers
00357         core_cmd = ['patch', 'circ', self.conf_mat_ID, *self.discr_core, 0, 0, 0, self.r_core]
00358 
00359         # Create the concrete cover fibers
00360         cover_cmd = ['patch', 'circ', self.unconf_mat_ID, *self.discr_cover, 0, 0, self.r_core, self.b/2]
00361         self.fib_sec = [['section', 'Fiber', self.ID, '-GJ', self.GJ],
00362             core_cmd, cover_cmd]
00363 
00364         # Create the reinforcing fibers
00365         bars_cmd = ['layer', 'circ', self.bars_mat_ID, self.n_bars, self.Ay, 0, 0, self.r_bars, self.alpha_i]
00366         self.fib_sec.append(bars_cmd)
00367 
00368         # Data storage for loading/saving
00369         self.UpdateStoredData()
00370 
00371 
```

7.10.3.3 ShowInfo()

```
def ShowInfo (
        self,
        plot = False,
        block = False )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the material model. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 399 of file Fibers.py.

```
00399     def ShowInfo(self, plot = False, block = False):
00400         """
00401         Implementation of the homonym abstract method.
```

```
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        """
        print("")
        print("Requested info for FibersCirc, ID = {}".format(self.ID))
        print("Section associated: {} ".format(self.section_name_tag))
        print("Base b = {} mm and concrete cover e = {} mm".format(self.b/mm_unit, self.e/mm_unit))
        print("Radius of the confined core r_core = {} mm, radius of the bars range r_bars = {} mm and initial angle alpha_i = {} deg".format(self.r_core/mm_unit, self.r_bars/mm_unit, self.alpha_i))
        print("Confined material model ID = {}".format(self.conf_mat_ID))
        print("Unconfined material model ID = {}".format(self.unconf_mat_ID))
        print("Bars material model ID = {}".format(self.bars_mat_ID))
        print("Discretisation in the core [number of wedges, number of rings] = {}".format(self.discr_core))
        print("Discretisation in the lateral covers [number of wedges, number of rings] = {}".format(self.discr_cover))
        print("")

        if plot:
            plot_fiber_section(self.fib_sec, matcolor=['#808080', '#D3D3D3', 'k'])

            if block:
                plt.show()


```

7.10.3.4 UpdateStoredData()

```
def UpdateStoredData (
                self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 373 of file Fibers.py.

```
00373     def UpdateStoredData(self):
00374         """
00375         Implementation of the homonym abstract method.
00376         See parent class DataManagement for detailed information.
00377         """
00378         self.data = [["INFO_TYPE", "FibersCirc"], # Tag for differentiating different data
00379             ["ID", self.ID],
00380             ["section_name_tag", self.section_name_tag],
00381             ["b", self.b],
00382             ["e", self.e],
00383             ["r_core", self.r_core],
00384             ["D_bars", self.D_bars],
00385             ["Ay", self.Ay],
00386             ["n_bars", self.n_bars],
00387             ["r_bars", self.r_bars],
00388             ["D_hoops", self.D_hoops],
00389             ["alpha_i", self.alpha_i],
00390             ["GJ", self.GJ],
00391             ["conf_mat_ID", self.conf_mat_ID],
00392             ["discr_core", self.discr_core],
00393             ["unconf_mat_ID", self.unconf_mat_ID],
00394             ["discr_cover", self.discr_cover],
00395             ["bars_mat_ID", self.bars_mat_ID],
00396             ["Initialized", self.Initialized]]
00397
00398
```

7.10.4 Member Data Documentation

7.10.4.1 alpha_i

`alpha_i`

Definition at line 336 of file Fibers.py.

7.10.4.2 Ay

`Ay`

Definition at line 328 of file Fibers.py.

7.10.4.3 b

`b`

Definition at line 325 of file Fibers.py.

7.10.4.4 bars_mat_ID

`bars_mat_ID`

Definition at line 333 of file Fibers.py.

7.10.4.5 conf_mat_ID

`conf_mat_ID`

Definition at line 332 of file Fibers.py.

7.10.4.6 D_bars

`D_bars`

Definition at line 327 of file Fibers.py.

7.10.4.7 D_hoops

```
D_hoops
```

Definition at line 330 of file Fibers.py.

7.10.4.8 data

```
data
```

Definition at line 378 of file Fibers.py.

7.10.4.9 discr_core

```
discr_core
```

Definition at line 334 of file Fibers.py.

7.10.4.10 discr_cover

```
discr_cover
```

Definition at line 335 of file Fibers.py.

7.10.4.11 e

```
e
```

Definition at line 326 of file Fibers.py.

7.10.4.12 fib_sec

```
fib_sec
```

Definition at line 361 of file Fibers.py.

7.10.4.13 GJ

```
GJ
```

Definition at line 337 of file Fibers.py.

7.10.4.14 ID

```
ID
```

Definition at line 324 of file Fibers.py.

7.10.4.15 Initialized

```
Initialized
```

Definition at line 341 of file Fibers.py.

7.10.4.16 n_bars

```
n_bars
```

Definition at line 329 of file Fibers.py.

7.10.4.17 r_bars

```
r_bars
```

Definition at line 353 of file Fibers.py.

7.10.4.18 r_core

```
r_core
```

Definition at line 354 of file Fibers.py.

7.10.4.19 section_name_tag

`section_name_tag`

Definition at line 340 of file Fibers.py.

7.10.4.20 unconf_mat_ID

`unconf_mat_ID`

Definition at line 331 of file Fibers.py.

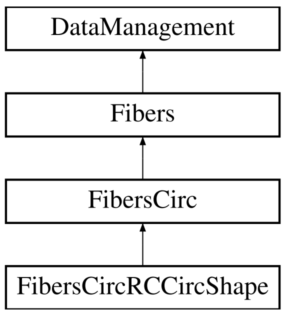
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers.py

7.11 FibersCircRCCircShape Class Reference

Class that is the children of FibersCirc and combine the class RCCircShape (section) to retrieve the information needed.

Inheritance diagram for FibersCircRCCircShape:



Public Member Functions

- def __init__ (self, int ID, RCCircShape section, int unconf_mat_ID, int conf_mat_ID, int bars_mat_ID, list discr_core, list discr_cover, alpha_i=0.0, GJ=0)

    Constructor of the class.

Public Attributes

- section
- section_name_tag

7.11.1 Detailed Description

Class that is the children of FibersCirc and combine the class RCCircShape (section) to retrieve the information needed.

Parameters

FibersCirc	Parent class.

Definition at line 435 of file Fibers.py.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 __init__()

```
def __init__ (
              self,
             int ID,
             RCCircShape section,
             int unconf_mat_ID,
             int conf_mat_ID,
             int bars_mat_ID,
             list discr_core,
             list discr_cover,
              alpha_i = 0.0,
              GJ = 0 )
```

Constructor of the class.

Parameters

ID	(int): Unique fiber section ID.
section	(RCCircShape): RCCircShape section object.
unconf_mat_ID	(int): ID of material model that will be assigned to the unconfined fibers.
conf_mat_ID	(int): ID of material model that will be assigned to the confined fibers.
bars_mat_ID	(int): ID of material model that will be assigned to the reinforcing bars fibers.
discr_core	(list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges), number of subdivisions (fibers) in the radial direction (number of rings) for the confined core.
discr_cover	(list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges), number of subdivisions (fibers) in the radial direction (number of rings) for the unconfined cover.
alpha_i	(float, optional): Angle in deg of the first vertical rebars with respect to the y axis, counterclockwise. Defaults to 0.0.
GJ	(float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

Reimplemented from FibersCirc.

Definition at line 441 of file Fibers.py.

```
            discr_core: list, discr_cover: list, alpha_i=0.0, GJ=0):
            """
            Constructor of the class.

            @param ID (int): Unique fiber section ID.
```

```
00447         @param section (RCCircShape): RCCircShape section object.
00448         @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined fibers.
00449         @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
00450         @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars fibers.
00451         @param discr_core (list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges),
00452             number of subdivisions (fibers) in the radial direction (number of rings) for the confined core.
00453         @param discr_cover (list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges),
00454             number of subdivisions (fibers) in the radial direction (number of rings) for the unconfined cover.
00455         @param alpha_i (float, optional): Angle in deg of the first vertical rebars with respect to the y axis, counterclockwise. Defaults to 0.0.
00456         @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.
00457         """
00458         self.section = deepcopy(section)
00459         super().__init__(ID, section.b, section.e, section.D_bars, section.Ay, section.n_bars, section.D_hoops, unconf_mat_ID, conf_mat_ID, bars_mat_ID,
00460             discr_core, discr_cover, alpha_i=alpha_i, GJ=GJ)
00461         self.section_name_tag = section.name_tag
00462         self.UpdateStoredData()
00463
00464
```

7.11.3 Member Data Documentation

7.11.3.1 section

`section`

Definition at line 458 of file Fibers.py.

7.11.3.2 section_name_tag

`section_name_tag`

Definition at line 461 of file Fibers.py.

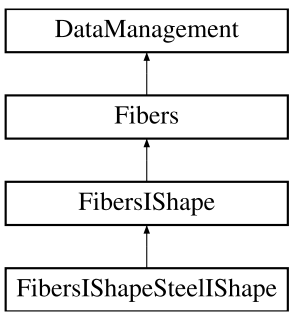
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers.py

7.12 FibersIShape Class Reference

Class that stores funcions, material properties, geometric and mechanical parameters for a steel I shape (non double symmetric) fiber section.

Inheritance diagram for FibersIShape:



Public Member Functions

- def __init__ (self, int ID, d, bf_t, bf_b, tf_t, tf_b, tw, int top_flange_mat_ID, int bottom_flange_mat_ID, int web_mat_ID, list discr_top_flange, list discr_bottom_flange, list discr_web, GJ=0.0)

  Constructor of the class.
- def CreateFibers (self)

  Method that initialise the fiber by calling the OpenSeesPy commands.
- def ReInit (self)

  Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False)

  Implementation of the homonym abstract method.
- def UpdateStoredData (self)

  Implementation of the homonym abstract method.

Public Attributes

- bf_b
- bf_t
- bottom_flange_mat_ID
- d
- data
- discr_bottom_flange
- discr_top_flange
- discr_web
- fib_sec
- GJ
- ID
- Initialized
- section_name_tag
- tf_b
- tf_t
- top_flange_mat_ID
- tw
- web_mat_ID

7.12.1 Detailed Description

Class that stores funcions, material properties, geometric and mechanical parameters for a steel I shape (non double symmetric) fiber section.

Coordinates: plotting coordinte (x, y) = fiber section coordinate (z, y) = (-x, y). For more information, see the OpenSeesPy documentation.

Parameters

Fibers	Parent abstract class.

Definition at line 465 of file Fibers.py.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 __init__()

```
def __init__ (
              self,
             int ID,
              d,
              bf_t,
              bf_b,
              tf_t,
              tf_b,
              tw,
             int top_flange_mat_ID,
             int bottom_flange_mat_ID,
             int web_mat_ID,
             list discr_top_flange,
             list discr_bottom_flange,
             list discr_web,
              GJ = 0.0 )
```

Constructor of the class.

Parameters

ID	(int): Unique fiber section ID.
d	(float): Depth of the section.
bf_t	(float): Top flange's width of the section
bf_b	(float): Bottom flange's width of the section
tf_t	(float): Top flange's thickness of the section
tf_b	(float): Bottom flange's thickness of the section
tw	(float): Web's thickness of the section
top_flange_mat_ID	(int): ID of material model that will be assigned to the top flange fibers.
bottom_flange_mat_ID	(int): ID of material model that will be assigned to the bottom flange fibers.
web_mat_ID	(int): ID of material model that will be assigned to the web fibers.
discr_top_flange	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top flange.
discr_bottom_flange	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the bottom flange.
discr_web	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the web.
GJ	(float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	d needs to be positive.
NegativeValue	bf_t needs to be positive.
NegativeValue	bf_b needs to be positive.
NegativeValue	tf_t needs to be positive.
NegativeValue	tf_b needs to be positive.

Exceptions

NegativeValue	tw needs to be positive.
NegativeValue	top_flange_mat_ID needs to be a positive integer.
NegativeValue	bottom_flange_mat_ID needs to be a positive integer.
NegativeValue	web_mat_ID needs to be a positive integer.
WrongDimension	discr_top_flange has a length of 2.
WrongDimension	discr_bottom_flange has a length of 2.
WrongDimension	discr_web has a length of 2.
NegativeValue	GJ needs to be positive.
InconsistentGeometry	The sum of the flanges thickness can't be bigger than d.

Reimplemented in FibersIShapeSteelIShape.

Definition at line 472 of file Fibers.py.

```
        discr_top_flange: list, discr_bottom_flange: list, discr_web: list,  GJ = 0.0):
        """
        Constructor of the class.

        @param ID (int): Unique fiber section ID.
        @param d (float): Depth of the section.
        @param bf_t (float): Top flange’s width of the section
        @param bf_b (float): Bottom flange’s width of the section
        @param tf_t (float): Top flange’s thickness of the section
        @param tf_b (float): Bottom flange’s thickness of the section
        @param tw (float): Web’s thickness of the section
        @param top_flange_mat_ID (int): ID of material model that will be assigned to the top flange fibers.
        @param bottom_flange_mat_ID (int): ID of material model that will be assigned to the bottom flange fibers.
        @param web_mat_ID (int): ID of material model that will be assigned to the web fibers.
        @param discr_top_flange (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top flange.
        @param discr_bottom_flange (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the bottom flange.
        @param discr_web (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the web.
        @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: d needs to be positive.
        @exception NegativeValue: bf_t needs to be positive.
        @exception NegativeValue: bf_b needs to be positive.
        @exception NegativeValue: tf_t needs to be positive.
        @exception NegativeValue: tf_b needs to be positive.
        @exception NegativeValue: tw needs to be positive.
        @exception NegativeValue: top_flange_mat_ID needs to be a positive integer.
        @exception NegativeValue: bottom_flange_mat_ID needs to be a positive integer.
        @exception NegativeValue: web_mat_ID needs to be a positive integer.
        @exception WrongDimension: discr_top_flange has a length of 2.
        @exception WrongDimension: discr_bottom_flange has a length of 2.
        @exception WrongDimension: discr_web has a length of 2.
        @exception NegativeValue: GJ needs to be positive.
        @exception InconsistentGeometry: The sum of the flanges thickness can’t be bigger than d.
        """
        # Check
        if ID < 1: raise NegativeValue()
        if d < 0: raise NegativeValue()
        if bf_t < 0: raise NegativeValue()
        if bf_b < 0: raise NegativeValue()
        if tf_b < 0: raise NegativeValue()
        if tf_t < 0: raise NegativeValue()
        if tw < 0: raise NegativeValue()
        if top_flange_mat_ID < 1: raise NegativeValue()
        if bottom_flange_mat_ID < 1: raise NegativeValue()
        if web_mat_ID < 1: raise NegativeValue()
        if len(discr_top_flange) != 2: raise WrongDimension()
        if len(discr_bottom_flange) != 2: raise WrongDimension()
        if len(discr_web) != 2: raise WrongDimension()
        if GJ < 0: raise NegativeValue()
        if tf_t+tf_b >= d: raise InconsistentGeometry()

        # Arguments
        self.ID = ID
        self.d = d
```

```
        self.bf_t = bf_t
        self.bf_b = bf_b
        self.tf_t = tf_t
        self.tf_b = tf_b
        self.tw = tw
        self.top_flange_mat_ID = top_flange_mat_ID
        self.bottom_flange_mat_ID = bottom_flange_mat_ID
        self.web_mat_ID = web_mat_ID
        self.discr_top_flange = copy(discr_top_flange)
        self.discr_bottom_flange = copy(discr_bottom_flange)
        self.discr_web = copy(discr_web)
        self.GJ = GJ

        # Initialized the parameters that are dependent from others
        self.section_name_tag = "None"
        self.Initialized = False
        self.ReInit()

```

7.12.3 Member Function Documentation

7.12.3.1 CreateFibers()

```
def CreateFibers (
            self )
```

Method that initialise the fiber by calling the OpenSeesPy commands.

Definition at line 631 of file Fibers.py.

```
00631     def CreateFibers(self):
00632         """
00633         Method that initialise the fiber by calling the OpenSeesPy commands.
00634         """
00635         create_fiber_section(self.fib_sec)
00636         self.Initialized = True
00637         self.UpdateStoredData()
00638
00639
```

7.12.3.2 ReInit()

```
def ReInit (
            self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 546 of file Fibers.py.

```
00546     def ReInit(self):
00547         """
00548         Implementation of the homonym abstract method.
00549         See parent class DataManagement for detailed information.
00550         """
00551         # Memebers
00552         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
      (modified)"
00553
00554         # Parameters
00555         z1 = self.tw/2
00556         y1 = (self.d - self.tf_t - self.tf_b)/2
00557
```

```
        # Create the flange top
        flange_top = [y1, -self.bf_t/2, y1+self.tf_t, self.bf_t/2]
        flange_top_cmd = ['patch', 'rect', self.top_flange_mat_ID, *self.discr_top_flange,
*flange_top]

        # Create the flange bottom
        flange_bottom = [-y1-self.tf_b, -self.bf_b/2, -y1, self.bf_b/2]
        flange_bottom_cmd = ['patch', 'rect', self.bottom_flange_mat_ID, *self.discr_bottom_flange,
*flange_bottom]

        # Create the web
        web = [-y1, -z1, y1, z1]
        web_cmd = ['patch', 'rect', self.web_mat_ID, *self.discr_web, *web]

        self.fib_sec = [['section', 'Fiber', self.ID, '-GJ', self.GJ],
            flange_top_cmd, web_cmd, flange_bottom_cmd]

        # Data storage for loading/saving
        self.UpdateStoredData()


```

7.12.3.3 ShowInfo()

```
def ShowInfo (
        self,
        plot = False,
        block = False )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the fiber. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 602 of file Fibers.py.

```
00602     def ShowInfo(self, plot = False, block = False):
00603         """
00604         Implementation of the homonym abstract method.
00605         See parent class DataManagement for detailed information.
00606 
00607         @param plot (bool, optional): Option to show the plot of the fiber. Defaults to False.
00608         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the program everytime that a plot should pop up). Defaults to False.
00609         """
00610         print("")
00611         print("Requested info for FibersRect, ID = {}".format(self.ID))
00612         print("Section associated: {} ".format(self.section_name_tag))
00613         print("Depth d = {} mm and web thickness tw = {} mm".format(self.d/mm_unit, self.tw/mm_unit))
00614         print("Top flange width bf_t = {} mm and thickness tf_t = {} mm".format(self.bf_t/mm_unit,
self.tf_t/mm_unit))
00615         print("Bottom flange width bf_b = {} mm and thickness tf_b = {} mm".format(self.bf_b/mm_unit,
self.tf_b/mm_unit))
00616         print("Web material model ID = {}".format(self.web_mat_ID))
00617         print("Top flange material model ID = {}".format(self.top_flange_mat_ID))
00618         print("Bottom flange material model ID = {}".format(self.bottom_flange_mat_ID))
00619         print("Discretisation in the web [IJ or x/z dir, JK or y dir] = {}".format(self.discr_web))
00620         print("Discretisation in the top flange [IJ or x/z dir, JK or y dir] =
{}".format(self.discr_top_flange))
00621         print("Discretisation in the bottom flange [IJ or x/z dir, JK or y dir] =
{}".format(self.discr_bottom_flange))
00622         print("")
00623 
00624         if plot:
00625             plot_fiber_section(self.fib_sec, matcolor=['r', 'b', 'g', 'k'])
```

```
            if block:
                plt.show()


```

7.12.3.4 UpdateStoredData()

```
def UpdateStoredData (
                self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 578 of file Fibers.py.

```
00578     def UpdateStoredData(self):
00579         """
00580         Implementation of the homonym abstract method.
00581         See parent class DataManagement for detailed information.
00582         """
00583         self.data = [["INFO_TYPE", "FibersIShape"], # Tag for differentiating different data
00584             ["ID", self.ID],
00585             ["section_name_tag", self.section_name_tag],
00586             ["d", self.d],
00587             ["bf_t", self.bf_t],
00588             ["bf_b", self.bf_b],
00589             ["tf_t", self.tf_t],
00590             ["tf_b", self.tf_b],
00591             ["tw", self.tw],
00592             ["GJ", self.GJ],
00593             ["top_flange_mat_ID", self.top_flange_mat_ID],
00594             ["bottom_flange_mat_ID", self.bottom_flange_mat_ID],
00595             ["web_mat_ID", self.web_mat_ID],
00596             ["discr_top_flange", self.discr_top_flange],
00597             ["discr_bottom_flange", self.discr_bottom_flange],
00598             ["discr_web", self.discr_web],
00599             ["Initialized", self.Initialized]]
00600
00601
```

7.12.4 Member Data Documentation

7.12.4.1 bf_b

```
bf_b
```

Definition at line 529 of file Fibers.py.

7.12.4.2 bf_t

```
bf_t
```

Definition at line 528 of file Fibers.py.

7.12.4.3 bottom_flange_mat_ID

```
bottom_flange_mat_ID
```

Definition at line 534 of file Fibers.py.

7.12.4.4 d

```
d
```

Definition at line 527 of file Fibers.py.

7.12.4.5 data

```
data
```

Definition at line 583 of file Fibers.py.

7.12.4.6 discr_bottom_flange

```
discr_bottom_flange
```

Definition at line 537 of file Fibers.py.

7.12.4.7 discr_top_flange

```
discr_top_flange
```

Definition at line 536 of file Fibers.py.

7.12.4.8 discr_web

```
discr_web
```

Definition at line 538 of file Fibers.py.

7.12.4.9 fib_sec

`fib_sec`

Definition at line 570 of file Fibers.py.

7.12.4.10 GJ

`GJ`

Definition at line 539 of file Fibers.py.

7.12.4.11 ID

`ID`

Definition at line 526 of file Fibers.py.

7.12.4.12 Initialized

`Initialized`

Definition at line 543 of file Fibers.py.

7.12.4.13 section_name_tag

`section_name_tag`

Definition at line 542 of file Fibers.py.

7.12.4.14 tf_b

`tf_b`

Definition at line 531 of file Fibers.py.

7.12.4.15 tf_t

```
tf_t
```

Definition at line 530 of file Fibers.py.

7.12.4.16 top_flange_mat_ID

```
top_flange_mat_ID
```

Definition at line 533 of file Fibers.py.

7.12.4.17 tw

```
tw
```

Definition at line 532 of file Fibers.py.

7.12.4.18 web_mat_ID

```
web_mat_ID
```

Definition at line 535 of file Fibers.py.

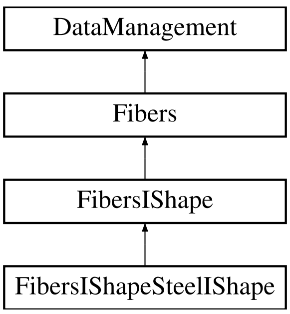
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers.py

7.13 FibersIShapeSteelIShape Class Reference

Class that is the children of FibersIShape and combine the class SteelIShape (section) to retrieve the information needed.

Inheritance diagram for FibersIShapeSteelIShape:



Public Member Functions

- def __init__ (self, int ID, SteelIShape section, int top_flange_mat_ID, list discr_top_flange, list discr_bottom_flange, list discr_web, GJ=0.0, bottom_flange_mat_ID=-1, web_mat_ID=-1)

  Constructor of the class.

Public Attributes

- section
- section_name_tag

7.13.1 Detailed Description

Class that is the children of FibersIShape and combine the class SteelIShape (section) to retrieve the information needed.

Parameters

FibersIShape	Parent class.

Definition at line 640 of file Fibers.py.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 __init__()

```
def __init__ (
              self,
             int ID,
             SteelIShape section,
             int top_flange_mat_ID,
             list discr_top_flange,
             list discr_bottom_flange,
             list discr_web,
              GJ = 0.0,
              bottom_flange_mat_ID = -1,
              web_mat_ID = -1 )
```

Constructor of the class.

Parameters

ID	(int): Unique fiber section ID.
section	(SteelIShape): SteelIShape section object.

Parameters

top_flange_mat_ID	(int): ID of material model that will be assigned to the top flange fibers.
discr_top_flange	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top flange.
discr_bottom_flange	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the bottom flange.
discr_web	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the web.
GJ	(float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.
bottom_flange_mat_ID	(int): ID of material model that will be assigned to the bottom flange fibers. Defaults to -1, e.g. equal to top_flange_mat_ID.
web_mat_ID	(int): ID of material model that will be assigned to the web fibers. Defaults to -1, e.g. equal to top_flange_mat_ID.

Reimplemented from FibersIShape.

Definition at line 646 of file Fibers.py.

```
00647         GJ=0.0, bottom_flange_mat_ID = -1, web_mat_ID = -1):
00648         """
00649         Constructor of the class.
00650 
00651         @param ID (int): Unique fiber section ID.
00652         @param section (SteelIShape): SteelIShape section object.
00653         @param top_flange_mat_ID (int): ID of material model that will be assigned to the top flange fibers.
00654         @param discr_top_flange (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top flange.
00655         @param discr_bottom_flange (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the bottom flange.
00656         @param discr_web (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the web.
00657         @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.
00658         @param bottom_flange_mat_ID (int): ID of material model that will be assigned to the bottom flange fibers.
00659             Defaults to -1, e.g. equal to top_flange_mat_ID.
00660         @param web_mat_ID (int): ID of material model that will be assigned to the web fibers.
00661             Defaults to -1, e.g. equal to top_flange_mat_ID.
00662         """
00663         self.section = deepcopy(section)
00664         if bottom_flange_mat_ID == -1: bottom_flange_mat_ID = top_flange_mat_ID
00665         if web_mat_ID == -1: web_mat_ID = top_flange_mat_ID
00666 
00667         super().__init__(ID, section.d, section.bf, section.bf, section.tf, section.tf, section.tw, top_flange_mat_ID, bottom_flange_mat_ID, web_mat_ID,
00668             discr_top_flange, discr_bottom_flange, discr_web, GJ)
00669         self.section_name_tag = section.name_tag
00670         self.UpdateStoredData()
00671 
00672 
```

7.13.3 Member Data Documentation

7.13.3.1 section

```
section
```

Definition at line 663 of file Fibers.py.

7.13.3.2 section_name_tag

```
section_name_tag
```

Definition at line 669 of file Fibers.py.

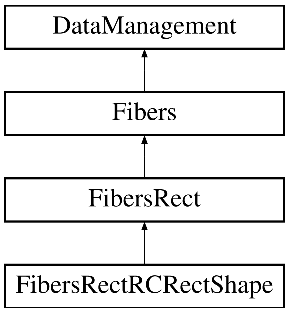
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers.py

7.14 FibersRect Class Reference

Class that stores funcions, material properties, geometric and mechanical parameters for a rectangular RC fiber section.

Inheritance diagram for FibersRect:



Public Member Functions

- def __init__ (self, int ID, b, d, Ay, D_hoops, e, int unconf_mat_ID, int conf_mat_ID, int bars_mat_ID, np.↩ ndarray bars_x, np.ndarray ranges_y, list discr_core, list discr_cover_lateral, list discr_cover_topbottom, GJ=0.0)

  Constructor of the class.
- def CreateFibers (self)

  Method that initialises the fiber by calling the OpenSeesPy commands.
- def ReInit (self)

  Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False)

  Implementation of the homonym abstract method.
- def UpdateStoredData (self)

  Implementation of the homonym abstract method.

Public Attributes

- Ay
- b
- bars_mat_ID
- bars_x
- conf_mat_ID
- d
- D_hoops
- data
- discr_core
- discr_cover_lateral
- discr_cover_topbottom
- e
- fib_sec
- GJ
- ID
- Initialized
- ranges_y
- rebarYZ
- section_name_tag
- unconf_mat_ID

7.14.1 Detailed Description

Class that stores funcions, material properties, geometric and mechanical parameters for a rectangular RC fiber section.

Coordinates: plotting coordinte (x, y) = fiber section coordinate (z, y) = (-x, y). For more information, see the OpenSeesPy documentation.

Parameters

Fibers	Parent abstract class.

Definition at line 28 of file Fibers.py.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 __init__()

```
def __init__ (
              self,
              int ID,
              b,
              d,
              Ay,
```

```
 D_hoops,
 e,
int unconf_mat_ID,
int conf_mat_ID,
int bars_mat_ID,
np.ndarray bars_x,
np.ndarray ranges_y,
list discr_core,
list discr_cover_lateral,
list discr_cover_topbottom,
 GJ = 0.0 )
```

Constructor of the class.

Parameters

ID	(int): Unique fiber section ID.
b	(float): Width of the section.
d	(float): Depth of the section.
Ay	(float): Area of one vertical reinforcing bar.
D_hoops	(float): Diameter of the hoops.
e	(float): Concrete cover.
unconf_mat_ID	(int): ID of material model that will be assigned to the unconfined fibers.
conf_mat_ID	(int): ID of material model that will be assigned to the confined fibers.
bars_mat_ID	(int): ID of material model that will be assigned to the reinforcing bars fibers.
bars_x	(np.ndarray): Array with a range of aligned vertical reinforcing bars for each row in x direction. Distances from border to bar centerline, bar to bar centerlines and finally bar centerline to border in the x direction (aligned). Starting from the left to right, from the top range to the bottom one. The number of bars for each range can vary; in this case, add this argument when defining the array " dtype = object"
ranges_y	(np.ndarray): Array of dimension 1 with the position or spacing in y of the ranges in bars_x. Distances from border to range centerlines, range to range centerlines and finally range centerline to border in the y direction. Starting from the top range to the bottom one.
discr_core	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the confined core.
discr_cover_lateral	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the lateral unconfined cover.
discr_cover_topbottom	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top and bottom unconfined cover.
GJ	(float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	b needs to be positive.
NegativeValue	d needs to be positive.
NegativeValue	Ay needs to be positive.
NegativeValue	D_hoops needs to be positive.
NegativeValue	e needs to be positive.
NegativeValue	unconf_mat_ID needs to be a positive integer.
NegativeValue	conf_mat_ID needs to be a positive integer.
NegativeValue	bars_mat_ID needs to be a positive integer.

Exceptions

WrongDimension	Number of rows in the list bars_x needs to be the same of the length of ranges_y - 1.
InconsistentGeometry	The sum of the distances for each row in bars_x should be equal to the section's width (tol = 5 mm).
InconsistentGeometry	The sum of the distances in ranges_y should be equal to the section's depth (tol = 5 mm).
InconsistentGeometry	e should be smaller than half the depth and the width of the section.
WrongDimension	discr_core has a length of 2.
WrongDimension	discr_cover_lateral has a length of 2.
WrongDimension	discr_cover_topbottom has a length of 2.
NegativeValue	GJ needs to be positive.

Reimplemented in FibersRectRCRectShape.

Definition at line 35 of file Fibers.py.

```
        bars_x: np.ndarray, ranges_y: np.ndarray, discr_core: list, discr_cover_lateral: list, discr_cover_topbottom: list, GJ = 0.0):
        """
        Constructor of the class.

        @param ID (int): Unique fiber section ID.
        @param b (float): Width of the section.
        @param d (float): Depth of the section.
        @param Ay (float): Area of one vertical reinforcing bar.
        @param D_hoops (float): Diameter of the hoops.
        @param e (float): Concrete cover.
        @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined fibers.
        @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
        @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars fibers.
        @param bars_x (np.ndarray): Array with a range of aligned vertical reinforcing bars for each row in x direction.
              Distances from border to bar centerline, bar to bar centerlines and finally bar centerline to border in the x direction (aligned).
              Starting from the left to right, from the top range to the bottom one.
              The number of bars for each range can vary; in this case, add this argument when defining the array " dtype = object"
        @param ranges_y (np.ndarray): Array of dimension 1 with the position or spacing in y of the ranges in bars_x.
              Distances from border to range centerlines, range to range centerlines and finally range centerline to border in the y direction.
              Starting from the top range to the bottom one.
        @param discr_core (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the confined core.
        @param discr_cover_lateral (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the lateral unconfined cover.
        @param discr_cover_topbottom (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top and bottom unconfined cover.
        @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: b needs to be positive.
        @exception NegativeValue: d needs to be positive.
        @exception NegativeValue: Ay needs to be positive.
        @exception NegativeValue: D_hoops needs to be positive.
        @exception NegativeValue: e needs to be positive.
        @exception NegativeValue: unconf_mat_ID needs to be a positive integer.
        @exception NegativeValue: conf_mat_ID needs to be a positive integer.
        @exception NegativeValue: bars_mat_ID needs to be a positive integer.
        @exception WrongDimension: Number of rows in the list bars_x needs to be the same of the length of ranges_y - 1.
        @exception InconsistentGeometry: The sum of the distances for each row in bars_x should be equal to the section's width (tol = 5 mm).
        @exception InconsistentGeometry: The sum of the distances in ranges_y should be equal to the section's depth (tol = 5 mm).
        @exception InconsistentGeometry: e should be smaller than half the depth and the width of the section.
        @exception WrongDimension: discr_core has a length of 2.
        @exception WrongDimension: discr_cover_lateral has a length of 2.
        @exception WrongDimension: discr_cover_topbottom has a length of 2.
        @exception NegativeValue: GJ needs to be positive.
        """
```

```
        # Check
        if ID < 1: raise NegativeValue()
        if b < 0: raise NegativeValue()
        if d < 0: raise NegativeValue()
        if Ay < 0: raise NegativeValue()
        if D_hoops < 0: raise NegativeValue()
        if e < 0: raise NegativeValue()
        if unconf_mat_ID < 1: raise NegativeValue()
        if conf_mat_ID < 1: raise NegativeValue()
        if bars_mat_ID < 1: raise NegativeValue()
        if np.size(bars_x) != np.size(ranges_y)-1: raise WrongDimension()
        geometry_tol = 5*mm_unit
        for bars in bars_x:
            if abs(np.sum(bars) - b) > geometry_tol: raise InconsistentGeometry()
        if abs(np.sum(ranges_y) - d) > geometry_tol: raise InconsistentGeometry()
        if e > b/2 or e > d/2: raise InconsistentGeometry()
        if len(discr_core) != 2: raise WrongDimension()
        if len(discr_cover_lateral) != 2: raise WrongDimension()
        if len(discr_cover_topbottom) != 2: raise WrongDimension()
        if GJ < 0: raise NegativeValue()

        # Arguments
        self.ID = ID
        self.b = b
        self.d = d
        self.Ay = Ay
        self.D_hoops = D_hoops
        self.e = e
        self.unconf_mat_ID = unconf_mat_ID
        self.conf_mat_ID = conf_mat_ID
        self.bars_mat_ID = bars_mat_ID
        self.bars_x = deepcopy(bars_x)
        self.ranges_y = copy(ranges_y)
        self.discr_core = copy(discr_core)
        self.discr_cover_lateral = copy(discr_cover_lateral)
        self.discr_cover_topbottom = copy(discr_cover_topbottom)
        self.GJ = GJ

        # Initialized the parameters that are dependent from others
        self.section_name_tag = "None"
        self.Initialized = False
        self.ReInit()

```

7.14.3 Member Function Documentation

7.14.3.1 CreateFibers()

```
def CreateFibers (
                self )
```

Method that initialises the fiber by calling the OpenSeesPy commands.

Definition at line 226 of file Fibers.py.

```
00226     def CreateFibers(self):
00227         """
00228         Method that initialises the fiber by calling the OpenSeesPy commands.
00229         """
00230         create_fiber_section(self.fib_sec)
00231         self.Initialized = True
00232         self.UpdateStoredData()
00233
00234
```

7.14.3.2 ReInit()

```
def ReInit (
            self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 122 of file Fibers.py.

```
00122     def ReInit(self):
00123         """
00124         Implementation of the homonym abstract method.
00125         See parent class DataManagement for detailed information.
00126         """
00127         # Memebers
00128         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
      (modified)"
00129
00130         # Parameters
00131         z1 = self.b/2
00132         y1 = self.d/2
00133         zc = z1-self.e-self.D_hoops/2
00134         yc = y1-self.e-self.D_hoops/2
00135
00136         # Create the concrete core fibers
00137         core = [-yc, -zc, yc, zc]
00138         core_cmd = ['patch', 'rect', self.conf_mat_ID, *self.discr_core, *core]
00139
00140         # Create the concrete cover fibers (bottom left, top right)
00141         cover_up = [yc, -z1, y1, z1]
00142         cover_down = [-y1, -z1, -yc, z1]
00143         cover_left = [-yc, zc, yc, z1]
00144         cover_right = [-yc, -z1, yc, -zc]
00145         cover_up_cmd = ['patch', 'rect', self.unconf_mat_ID, *self.discr_cover_topbottom, *cover_up]
00146         cover_down_cmd = ['patch', 'rect', self.unconf_mat_ID, *self.discr_cover_topbottom,
      *cover_down]
00147         cover_left_cmd = ['patch', 'rect', self.unconf_mat_ID, *self.discr_cover_lateral, *cover_left]
00148         cover_right_cmd = ['patch', 'rect', self.unconf_mat_ID, *self.discr_cover_lateral,
      *cover_right]
00149         self.fib_sec = [['section', 'Fiber', self.ID, '-GJ', self.GJ],
00150             core_cmd, cover_up_cmd, cover_down_cmd, cover_left_cmd, cover_right_cmd]
00151
00152         # Create the reinforcing fibers (top, middle, bottom)
00153         # NB: note the order of definition of bars_x and ranges_y
00154         nr_bars = 0
00155         for range in self.bars_x:
00156             nr_bars += np.size(range)-1
00157         rebarY = -np.cumsum(self.ranges_y[0:-1]) + y1
00158         self.rebarYZ = np.zeros((nr_bars, 2))
00159
00160         iter = 0
00161         for ii, Y in enumerate(rebarY):
00162             rebarZ = -np.cumsum(self.bars_x[ii][0:-1]) + z1
00163             for Z in rebarZ:
00164                 self.rebarYZ[iter, :] = [Y, Z]
00165                 iter = iter + 1
00166
00167         for YZ in self.rebarYZ:
00168             self.fib_sec.append(['layer', 'bar', self.bars_mat_ID, self.Ay, *YZ])
00169
00170         # Data storage for loading/saving
00171         self.UpdateStoredData()
00172
00173
```

7.14.3.3 ShowInfo()

```
def ShowInfo (
            self,
            plot = False,
            block = False )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the fiber. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 199 of file Fibers.py.

```
00199     def ShowInfo(self, plot = False, block = False):
00200         """
00201         Implementation of the homonym abstract method.
00202         See parent class DataManagement for detailed information.
00203 
00204         @param plot (bool, optional): Option to show the plot of the fiber. Defaults to False.
00205         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
00206         """
00207         print("")
00208         print("Requested info for FibersRect, ID = {}".format(self.ID))
00209         print("Section associated: {} ".format(self.section_name_tag))
00210         print("Base b = {} mm and depth d = {} mm".format(self.b/mm_unit, self.d/mm_unit))
00211         print("Confined material model ID = {}".format(self.conf_mat_ID))
00212         print("Unconfined material model ID = {}".format(self.unconf_mat_ID))
00213         print("Bars material model ID = {}".format(self.bars_mat_ID))
00214         print("Discretisation in the core [IJ or x/z dir, JK or y dir] = {}".format(self.discr_core))
00215         print("Discretisation in the lateral covers [IJ or x/z dir, JK or y dir] = {}".format(self.discr_cover_lateral))
00216         print("Discretisation in the top and bottom covers [IJ or x/z dir, JK or y dir] = {}".format(self.discr_cover_topbottom))
00217         print("")
00218 
00219         if plot:
00220             plot_fiber_section(self.fib_sec, matcolor=['#808080', '#D3D3D3', 'k'])
00221 
00222             if block:
00223                 plt.show()
00224 
00225 
```

7.14.3.4 UpdateStoredData()

```
def UpdateStoredData (
                self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 175 of file Fibers.py.

```
00175     def UpdateStoredData(self):
00176         """
00177         Implementation of the homonym abstract method.
00178         See parent class DataManagement for detailed information.
00179         """
00180         self.data = [["INFO_TYPE", "FibersRect"], # Tag for differentiating different data
00181             ["ID", self.ID],
00182             ["section_name_tag", self.section_name_tag],
00183             ["b", self.b],
00184             ["d", self.d],
00185             ["Ay", self.Ay],
00186             ["D_hoops", self.D_hoops],
00187             ["e", self.e],
00188             ["GJ", self.GJ],
00189             ["conf_mat_ID", self.conf_mat_ID],
00190             ["discr_core", self.discr_core],
00191             ["unconf_mat_ID", self.unconf_mat_ID],
00192             ["discr_cover_topbottom", self.discr_cover_topbottom],
00193             ["discr_cover_lateral", self.discr_cover_lateral],
00194             ["bars_mat_ID", self.bars_mat_ID],
00195             ["bars_x", self.bars_x],
00196             ["ranges_y", self.ranges_y],
00197             ["Initialized", self.Initialized]]
00198 
```

7.14.4 Member Data Documentation

7.14.4.1 Ay

```
Ay
```

Definition at line 104 of file Fibers.py.

7.14.4.2 b

```
b
```

Definition at line 102 of file Fibers.py.

7.14.4.3 bars_mat_ID

```
bars_mat_ID
```

Definition at line 109 of file Fibers.py.

7.14.4.4 bars_x

```
bars_x
```

Definition at line 110 of file Fibers.py.

7.14.4.5 conf_mat_ID

```
conf_mat_ID
```

Definition at line 108 of file Fibers.py.

7.14.4.6 d

```
d
```

Definition at line 103 of file Fibers.py.

7.14.4.7 D_hoops

```
D_hoops
```

Definition at line 105 of file Fibers.py.

7.14.4.8 data

```
data
```

Definition at line 180 of file Fibers.py.

7.14.4.9 discr_core

```
discr_core
```

Definition at line 112 of file Fibers.py.

7.14.4.10 discr_cover_lateral

```
discr_cover_lateral
```

Definition at line 113 of file Fibers.py.

7.14.4.11 discr_cover_topbottom

```
discr_cover_topbottom
```

Definition at line 114 of file Fibers.py.

7.14.4.12 e

`e`

Definition at line 106 of file Fibers.py.

7.14.4.13 fib_sec

`fib_sec`

Definition at line 149 of file Fibers.py.

7.14.4.14 GJ

`GJ`

Definition at line 115 of file Fibers.py.

7.14.4.15 ID

`ID`

Definition at line 101 of file Fibers.py.

7.14.4.16 Initialized

`Initialized`

Definition at line 119 of file Fibers.py.

7.14.4.17 ranges_y

`ranges_y`

Definition at line 111 of file Fibers.py.

7.14.4.18 rebarYZ

```
rebarYZ
```

Definition at line 158 of file Fibers.py.

7.14.4.19 section_name_tag

```
section_name_tag
```

Definition at line 118 of file Fibers.py.

7.14.4.20 unconf_mat_ID

```
unconf_mat_ID
```

Definition at line 107 of file Fibers.py.

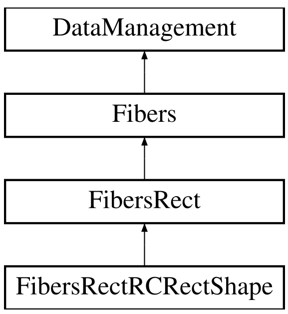
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers.py

7.15 FibersRectRCRectShape Class Reference

Class that is the children of FibersRect and combine the class RCRectShape (section) to retrieve the information needed.

Inheritance diagram for FibersRectRCRectShape:



Public Member Functions

- def __init__ (self, int ID, RCRectShape section, int unconf_mat_ID, int conf_mat_ID, int bars_mat_ID, list discr_core, list discr_cover_lateral, list discr_cover_topbottom, GJ=0)

  Constructor of the class.

Public Attributes

- section
- section_name_tag

7.15.1 Detailed Description

Class that is the children of FibersRect and combine the class RCRectShape (section) to retrieve the information needed.

Parameters

FibersRect	Parent class.

Definition at line 235 of file Fibers.py.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 __init__()

```
def __init__ (
        self,
        int ID,
        RCRectShape section,
        int unconf_mat_ID,
        int conf_mat_ID,
        int bars_mat_ID,
        list discr_core,
        list discr_cover_lateral,
        list discr_cover_topbottom,
        GJ = 0 )
```

Constructor of the class.

Parameters

ID	(int): Unique fiber section ID.
section	(RCRectShape): RCRectShape section object.
unconf_mat_ID	(int): ID of material model that will be assigned to the unconfined fibers.
conf_mat_ID	(int): ID of material model that will be assigned to the confined fibers.
bars_mat_ID	(int): ID of material model that will be assigned to the reinforcing bars fibers.
discr_core	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the confined core.
discr_cover_lateral	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the lateral unconfined core.
discr_cover_topbottom	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top and bottom unconfined core.
GJ	(float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

Reimplemented from FibersRect.

Definition at line 241 of file Fibers.py.

```
00242         discr_core: list, discr_cover_lateral: list, discr_cover_topbottom: list, GJ=0):
00243         """
00244         Constructor of the class.
00245 
00246         @param ID (int): Unique fiber section ID.
00247         @param section (RCRectShape): RCRectShape section object.
00248         @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined
       fibers.
00249         @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
00250         @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars
       fibers.
00251         @param discr_core (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the
       confined core.
00252         @param discr_cover_lateral (list): List with two entries: discretisation in IJ (x/z) and JK
       (y) for the lateral unconfined core.
00253         @param discr_cover_topbottom (list): List with two entries: discretisation in IJ (x/z) and JK
       (y) for the top and bottom unconfined core.
00254         @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section.
       Defaults to 0.0, assume no torsional stiffness.
00255         """
00256         self.section = deepcopy(section)
00257         super().__init__(ID, section.b, section.d, section.Ay, section.D_hoops, section.e,
       unconf_mat_ID, conf_mat_ID, bars_mat_ID,
00258             section.bars_position_x, section.bars_ranges_position_y, discr_core, discr_cover_lateral,
       discr_cover_topbottom, GJ=GJ)
00259         self.section_name_tag = section.name_tag
00260         self.UpdateStoredData()
00261 
00262 
```

7.15.3 Member Data Documentation

7.15.3.1 section

`section`

Definition at line 256 of file Fibers.py.

7.15.3.2 section_name_tag

`section_name_tag`

Definition at line 259 of file Fibers.py.

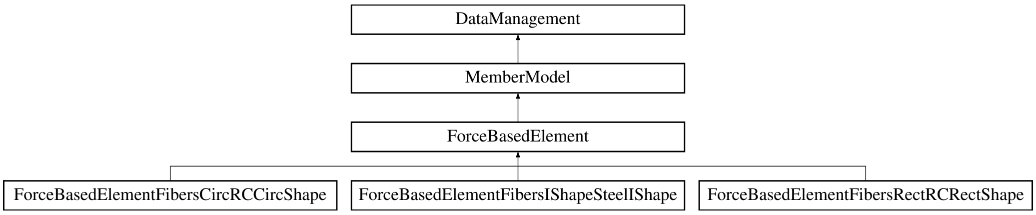
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers.py

7.16 ForceBasedElement Class Reference

Class that handles the storage and manipulation of a force-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Inheritance diagram for ForceBasedElement:



Public Member Functions

- def __init__ (self, int iNode_ID, int jNode_ID, int fiber_ID, int geo_transf_ID, new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION, tol=TOL_INTEGRATION, ele_ID=-1)

  Constructor of the class.
- def CreateMember (self)

  Method that initialises the member by calling the OpenSeesPy commands through various functions.
- def Record (self, str name_txt, str data_dir, force_rec=True, def_rec=True, time_rec=True)

  Implementation of the homonym abstract method.
- def RecordNodeDef (self, str name_txt, str data_dir, time_rec=True)

  Implementation of the homonym abstract method.
- def ReInit (self, new_integration_ID, ele_ID=-1)

  Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False)

  Implementation of the homonym abstract method.
- def UpdateStoredData (self)

  Implementation of the homonym abstract method.

Public Attributes

- data
- element_array
- element_ID
- fiber_ID
- geo_transf_ID
- Initialized
- iNode_ID
- integration_type
- Ip
- jNode_ID
- max_iter
- new_integration_ID
- section_name_tag
- tol

7.16.1 Detailed Description

Class that handles the storage and manipulation of a force-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Parameters

MemberModel	Parent abstract class.

Definition at line 964 of file MemberModel.py.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 __init__()

```
def __init__ (
              self,
             int iNode_ID,
             int jNode_ID,
             int fiber_ID,
             int geo_transf_ID,
              new_integration_ID = -1,
              Ip = 5,
              integration_type = "Lobatto",
              max_iter = MAX_ITER_INTEGRATION,
              tol = TOL_INTEGRATION,
              ele_ID = -1 )
```

Constructor of the class.

Parameters

iNode_ID	(int): ID of the first end node.
jNode_ID	(int): ID of the second end node.
fiber_ID	(int): ID of the fiber section.
geo_transf_ID	(int): The geometric transformation (for more information, see OpenSeesPy documentation).
new_integration_ID	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
Ip	(int, optional): Number of integration points (min. 3). Defaults to 5.
integration_type	(str, optional): Integration type. FOr more information, see OpenSeesPy documentation. Defaults to "Lobatto".
max_iter	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
tol	(float, optional): Tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	ID needs to be a positive integer.
NegativeValue	ID needs to be a positive integer.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	ID needs to be a positive integer, if different from -1.
NegativeValue	Ip needs to be a positive integer bigger than 3, if different from -1.
NegativeValue	max_iter needs to be a positive integer.
NegativeValue	tol needs to be positive.
NegativeValue	ID needs to be a positive integer, if different from -1.

Reimplemented in ForceBasedElementFibersCircRCCircShape, ForceBasedElementFibersIShapeSteelIShape, and ForceBasedElementFibersRectRCRectShape.

Definition at line 970 of file MemberModel.py.

```
        new_integration_ID = -1, Ip = 5, integration_type = "Lobatto", max_iter = MAX_ITER_INTEGRATION, tol = TOL_INTEGRATION, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param fiber_ID (int): ID of the fiber section.
        @param geo_transf_ID (int): The geometric transformation (for more information, see OpenSeesPy documentation).
        @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
        @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
        @param integration_type (str, optional): Integration type. FOr more information, see OpenSeesPy documentation.
            Defaults to "Lobatto".
        @param max_iter (int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
        @param tol (float, optional): Tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer, if different from -1.
        @exception NegativeValue: Ip needs to be a positive integer bigger than 3, if different from -1.
        @exception NegativeValue: max_iter needs to be a positive integer.
        @exception NegativeValue: tol needs to be positive.
        @exception NegativeValue: ID needs to be a positive integer, if different from -1.
        """
        # Check
        if iNode_ID < 1: raise NegativeValue()
        if jNode_ID < 1: raise NegativeValue()
        if fiber_ID < 1: raise NegativeValue()
        if geo_transf_ID < 1: raise NegativeValue()
        if new_integration_ID != -1 and new_integration_ID < 1: raise NegativeValue()
        if Ip != -1 and Ip < 3: raise NegativeValue()
        if max_iter < 0: raise NegativeValue()
        if tol < 0: raise NegativeValue()
        if ele_ID != -1 and ele_ID < 1: raise NegativeValue()

        # Arguments
        self.iNode_ID = iNode_ID
        self.jNode_ID = jNode_ID
        self.fiber_ID = fiber_ID
        self.geo_transf_ID = geo_transf_ID
        self.Ip = Ip
        self.integration_type = integration_type
        self.max_iter = max_iter
        self.tol = tol

        # Initialized the parameters that are dependent from others
        self.section_name_tag = "None"
        self.Initialized = False
        self.ReInit(new_integration_ID, ele_ID)


```

7.16.3 Member Function Documentation

7.16.3.1 CreateMember()

```
def CreateMember (
              self )
```

Method that initialises the member by calling the OpenSeesPy commands through various functions.

Definition at line 1092 of file MemberModel.py.

```
01092     def CreateMember(self):
01093         """
01094         Method that initialises the member by calling the OpenSeesPy commands through various functions.
01095         """
01096         self.element_array = [[self.element_ID, self.iNode_ID, self.jNode_ID]]
01097 
01098         # Define integration type
01099         beamIntegration(self.integration_type, self.new_integration_ID, self.fiber_ID, self.Ip)
01100 
01101         # Define element
01102         element('forceBeamColumn', self.element_ID, self.iNode_ID, self.jNode_ID, self.geo_transf_ID, self.new_integration_ID, '-iter', self.max_iter, self.tol)
01103 
01104         # Update class
01105         self.Initialized = True
01106         self.UpdateStoredData()
01107 
01108 
```

7.16.3.2 Record()

```
def Record (
              self,
             str name_txt,
             str data_dir,
              force_rec = True,
              def_rec = True,
              time_rec = True )
```

Implementation of the homonym abstract method.

See parent class MemberModel for detailed information.

Reimplemented from MemberModel.

Definition at line 1109 of file MemberModel.py.

```
01109     def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
01110         """
01111         Implementation of the homonym abstract method.
01112         See parent class MemberModel for detailed information.
01113         """
01114         super().Record(self.element_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec, time_rec=time_rec)
01115 
01116 
```

7.16.3.3 RecordNodeDef()

```
def RecordNodeDef (
        self,
        str name_txt,
        str data_dir,
        time_rec = True )
```

Implementation of the homonym abstract method.

See parent class MemberModel for detailed information.

Reimplemented from MemberModel.

Definition at line 1117 of file MemberModel.py.

```
01117     def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
01118         """
01119         Implementation of the homonym abstract method.
01120         See parent class MemberModel for detailed information.
01121         """
01122         super().RecordNodeDef(self.iNode_ID, self.jNode_ID, name_txt, data_dir, time_rec=time_rec)
01123 
01124 
```

7.16.3.4 ReInit()

```
def ReInit (
        self,
        new_integration_ID,
        ele_ID = -1 )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

new_integration_ID	(int): ID of the integration technique.
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Definition at line 1024 of file MemberModel.py.

```
01024     def ReInit(self, new_integration_ID, ele_ID = -1):
01025         """
01026         Implementation of the homonym abstract method.
01027         See parent class DataManagement for detailed information.
01028 
01029         @param new_integration_ID (int): ID of the integration technique.
01030         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
      IDConvention to define it.
01031         """
01032         # Precompute some members
01033         self.element_ID = IDConvention(self.iNode_ID, self.jNode_ID) if ele_ID == -1 else ele_ID
01034 
01035         # Arguments
01036         self.new_integration_ID = self.element_ID if new_integration_ID == -1 else new_integration_ID
01037 
01038         # Members
01039         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
      (modified)"
01040 
01041         # Data storage for loading/saving
```

```
        self.UpdateStoredData()


```

7.16.3.5 ShowInfo()

```
def ShowInfo (
            self,
            plot = False,
            block = False )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the material model. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 1066 of file MemberModel.py.

```
01066     def ShowInfo(self, plot = False, block = False):
01067         """
01068         Implementation of the homonym abstract method.
01069         See parent class DataManagement for detailed information.
01070 
01071         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
       False.
01072         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
       of the program everytime that a plot should pop up). Defaults to False.
01073         """
01074         print("")
01075         print("Requested info for ForceBasedElement member model, ID = {}".format(self.element_ID))
01076         print("Fiber associated, ID = {} ".format(self.fiber_ID))
01077         print("Integration type '{}', ID = {}".format(self.integration_type, self.new_integration_ID))
01078         print("Section associated {} ".format(self.section_name_tag))
01079         print("Number of integration points along the element Ip = {}, max iter = {}, tol =
       {}".format(self.Ip, self.max_iter, self.tol))
01080         print("Geometric transformation = {}".format(self.geo_transf_ID))
01081         print("")
01082 
01083         if plot:
01084             if self.Initialized:
01085                 plot_member(self.element_array, "ForceBased Element, ID = {}".format(self.element_ID))
01086                 if block:
01087                     plt.show()
01088             else:
01089                 print("The ForceBasedElement is not initialized (element not created), ID =
       {}".format(self.element_ID))
01090 
01091 
```

7.16.3.6 UpdateStoredData()

```
def UpdateStoredData (
            self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 1046 of file MemberModel.py.

```
01046     def UpdateStoredData(self):
01047         """
01048         Implementation of the homonym abstract method.
01049         See parent class DataManagement for detailed information.
01050         """
01051         self.data = [["INFO_TYPE", "ForceBasedElement"], # Tag for differentiating different data
01052             ["element_ID", self.element_ID],
01053             ["section_name_tag", self.section_name_tag],
01054             ["Ip", self.Ip],
01055             ["iNode_ID", self.iNode_ID],
01056             ["jNode_ID", self.jNode_ID],
01057             ["fiber_ID", self.fiber_ID],
01058             ["new_integration_ID", self.new_integration_ID],
01059             ["integration_type", self.integration_type],
01060             ["tol", self.tol],
01061             ["max_iter", self.max_iter],
01062             ["tranf_ID", self.geo_transf_ID],
01063             ["Initialized", self.Initialized]]
01064
01065
```

7.16.4 Member Data Documentation

7.16.4.1 data

`data`

Definition at line 1051 of file MemberModel.py.

7.16.4.2 element_array

`element_array`

Definition at line 1096 of file MemberModel.py.

7.16.4.3 element_ID

`element_ID`

Definition at line 1033 of file MemberModel.py.

7.16.4.4 fiber_ID

`fiber_ID`

Definition at line 1011 of file MemberModel.py.

7.16.4.5 geo_transf_ID

`geo_transf_ID`

Definition at line 1012 of file MemberModel.py.

7.16.4.6 Initialized

`Initialized`

Definition at line 1020 of file MemberModel.py.

7.16.4.7 iNode_ID

`iNode_ID`

Definition at line 1009 of file MemberModel.py.

7.16.4.8 integration_type

`integration_type`

Definition at line 1014 of file MemberModel.py.

7.16.4.9 Ip

`Ip`

Definition at line 1013 of file MemberModel.py.

7.16.4.10 jNode_ID

`jNode_ID`

Definition at line 1010 of file MemberModel.py.

7.16.4.11 max_iter

`max_iter`

Definition at line 1015 of file MemberModel.py.

7.16.4.12 new_integration_ID

`new_integration_ID`

Definition at line 1036 of file MemberModel.py.

7.16.4.13 section_name_tag

`section_name_tag`

Definition at line 1019 of file MemberModel.py.

7.16.4.14 tol

`tol`

Definition at line 1016 of file MemberModel.py.

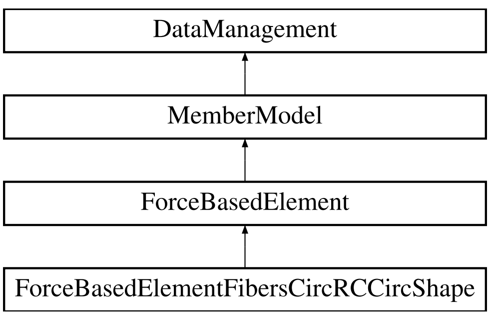
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.17 ForceBasedElementFibersCircRCCircShape Class Reference

Class that is the children of ForceBasedElement and combine the class FibersCircRCCircShape (fiber section) to retrieve the information needed.

Inheritance diagram for ForceBasedElementFibersCircRCCircShape:



Public Member Functions

- def __init__ (self, int iNode_ID, int jNode_ID, FibersCircRCCircShape fiber, int geo_transf_ID, new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION, tol=TOL_↩ INTEGRATION, ele_ID=-1)

  Constructor of the class.

Public Attributes

- section
- section_name_tag

7.17.1 Detailed Description

Class that is the children of ForceBasedElement and combine the class FibersCircRCCircShape (fiber section) to retrieve the information needed.

Parameters

ForceBasedElement	Parent class.

Definition at line 1157 of file MemberModel.py.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 __init__()

```
def __init__ (
              self,
             int iNode_ID,
             int jNode_ID,
             FibersCircRCCircShape fiber,
             int geo_transf_ID,
              new_integration_ID = -1,
              Ip = 5,
              integration_type = "Lobatto",
              max_iter = MAX_ITER_INTEGRATION,
              tol = TOL_INTEGRATION,
              ele_ID = -1 )
```

Constructor of the class.

Parameters

iNode_ID	(int): ID of the first end node.

Parameters

jNode_ID	(int): ID of the second end node.
fiber	(FibersCircRCCircShape): FibersCircRCCircShape fiber section object.
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
new_integration_ID	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
Ip	(int, optional): Number of integration points (min. 3). Defaults to 5.
integration_type	(str, optional): Integration type. FOr more information, see OpenSeesPy documentation. Defaults to "Lobatto".
max_iter	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
tol	(float, optional): Tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from ForceBasedElement.

Definition at line 1163 of file MemberModel.py.

```
01164         new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION,
      tol=TOL_INTEGRATION, ele_ID = -1):
01165         """
01166         Constructor of the class.
01167 
01168         @param iNode_ID (int): ID of the first end node.
01169         @param jNode_ID (int): ID of the second end node.
01170         @param fiber (FibersCircRCCircShape): FibersCircRCCircShape fiber section object.
01171         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
      documentation).
01172         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
      e.g. computed in ReInit().
01173         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01174         @param integration_type (str, optional): Integration type. FOr more information, see
      OpenSeesPy documentation.
01175             Defaults to "Lobatto".
01176         @param max_iter (int, optional): Maximal number of iteration to reach the integretion
      convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01177         @param tol (float, optional): Tolerance for the integration convergence. Defaults to
      TOL_INTEGRATION (Units).
01178         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
      IDConvention to define it.
01179         """
01180         self.section = deepcopy(fiber.section)
01181         super().__init__(iNode_ID, jNode_ID, fiber.ID, geo_transf_ID,
01182             new_integration_ID=new_integration_ID, Ip=Ip, integration_type=integration_type,
      max_iter=max_iter, tol=tol, ele_ID=ele_ID)
01183         self.section_name_tag = self.section.name_tag
01184         self.UpdateStoredData()
01185         # Check length
01186         self._CheckL()
01187 
01188 
```

7.17.3 Member Data Documentation

7.17.3.1 section

`section`

Definition at line 1180 of file MemberModel.py.

7.17.3.2 section_name_tag

`section_name_tag`

Definition at line 1183 of file MemberModel.py.

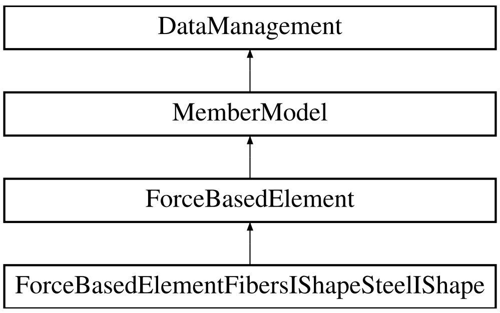
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.18 ForceBasedElementFibersIShapeSteelIShape Class Reference

Class that is the children of ForceBasedElement and combine the class FibersIShapeSteelIShape (fiber section) to retrieve the information needed.

Inheritance diagram for ForceBasedElementFibersIShapeSteelIShape:



Public Member Functions

- def __init__ (self, int iNode_ID, int jNode_ID, FibersIShapeSteelIShape fiber, int geo_transf_ID, new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION, tol=TOL_↩ INTEGRATION, ele_ID=-1)

    Constructor of the class.

Public Attributes

- section
- section_name_tag

7.18.1 Detailed Description

Class that is the children of ForceBasedElement and combine the class FibersIShapeSteelIShape (fiber section) to retrieve the information needed.

Parameters

ForceBasedElement	Parent class.

Definition at line 1189 of file MemberModel.py.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 __init__()

```
def __init__ (
                self,
               int iNode_ID,
               int jNode_ID,
               FibersIShapeSteelIShape fiber,
               int geo_transf_ID,
                new_integration_ID = -1,
                Ip = 5,
                integration_type = "Lobatto",
                max_iter = MAX_ITER_INTEGRATION,
                tol = TOL_INTEGRATION,
                ele_ID = -1 )
```

Constructor of the class.

Parameters

iNode_ID	(int): ID of the first end node.
jNode_ID	(int): ID of the second end node.
fiber	(FibersIShapeSteelIShape): FibersIShapeSteelIShape fiber section object.
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
new_integration_ID	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
Ip	(int, optional): Number of integration points (min. 3). Defaults to 5.
integration_type	(str, optional): Integration type. FOr more information, see OpenSeesPy documentation. Defaults to "Lobatto".
max_iter	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
tol	(float, optional): Tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from ForceBasedElement.

Definition at line 1195 of file MemberModel.py.

```
        new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION,
    tol=TOL_INTEGRATION, ele_ID = -1):
        """
```

```
01198         Constructor of the class.
01199
01200         @param iNode_ID (int): ID of the first end node.
01201         @param jNode_ID (int): ID of the second end node.
01202         @param fiber (FibersIShapeSteelIShape): FibersIShapeSteelIShape fiber section object.
01203         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
01204         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
e.g. computed in ReInit().
01205         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01206         @param integration_type (str, optional): Integration type. FOr more information, see
OpenSeesPy documentation.
01207             Defaults to "Lobatto".
01208         @param max_iter (int, optional): Maximal number of iteration to reach the integretion
convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01209         @param tol (float, optional): Tolerance for the integration convergence. Defaults to
TOL_INTEGRATION (Units).
01210         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
01211         """
01212         self.section = deepcopy(fiber.section)
01213         super().__init__(iNode_ID, jNode_ID, fiber.ID, geo_transf_ID,
01214             new_integration_ID=new_integration_ID, Ip=Ip, integration_type=integration_type,
max_iter=max_iter, tol=tol, ele_ID=ele_ID)
01215         self.section_name_tag = self.section.name_tag
01216         self.UpdateStoredData()
01217         # Check length
01218         self._CheckL()
01219
01220
```

7.18.3 Member Data Documentation

7.18.3.1 section

```
section
```

Definition at line 1212 of file MemberModel.py.

7.18.3.2 section_name_tag

```
section_name_tag
```

Definition at line 1215 of file MemberModel.py.

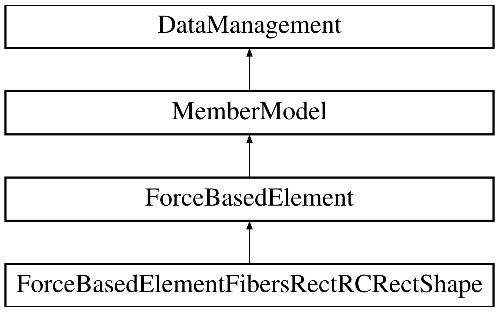
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.19 ForceBasedElementFibersRectRCRectShape Class Reference

Class that is the children of ForceBasedElement and combine the class FibersRectRCRectShape (fiber section) to retrieve the information needed.

Inheritance diagram for ForceBasedElementFibersRectRCRectShape:



Public Member Functions

- def __init__ (self, int iNode_ID, int jNode_ID, FibersRectRCRectShape fiber, int geo_transf_ID, new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION, tol=TOL_↩ INTEGRATION, ele_ID=-1)

  Constructor of the class.

Public Attributes

- section
- section_name_tag

7.19.1 Detailed Description

Class that is the children of ForceBasedElement and combine the class FibersRectRCRectShape (fiber section) to retrieve the information needed.

Parameters

ForceBasedElement	Parent class.

Definition at line 1125 of file MemberModel.py.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 __init__()

```
def __init__ (
        self,
        int iNode_ID,
        int jNode_ID,
        FibersRectRCRectShape fiber,
        int geo_transf_ID,
        new_integration_ID = -1,
        Ip = 5,
        integration_type = "Lobatto",
        max_iter = MAX_ITER_INTEGRATION,
        tol = TOL_INTEGRATION,
        ele_ID = -1 )
```

Constructor of the class.

Parameters

iNode_ID	(int): ID of the first end node.
jNode_ID	(int): ID of the second end node.
fiber	(FibersRectRCRectShape): FibersRectRCRectShape fiber section object.
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
new_integration_ID	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
Ip	(int, optional): Number of integration points (min. 3). Defaults to 5.
integration_type	(str, optional): Integration type. FOr more information, see OpenSeesPy documentation. Defaults to "Lobatto".
max_iter	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
tol	(float, optional): Tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from ForceBasedElement.

Definition at line 1131 of file MemberModel.py.

```
01132         new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION,
      tol=TOL_INTEGRATION, ele_ID = -1):
01133         """
01134         Constructor of the class.
01135
01136         @param iNode_ID (int): ID of the first end node.
01137         @param jNode_ID (int): ID of the second end node.
01138         @param fiber (FibersRectRCRectShape): FibersRectRCRectShape fiber section object.
01139         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
      documentation).
01140         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
      e.g. computed in ReInit().
01141         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01142         @param integration_type (str, optional): Integration type. FOr more information, see
      OpenSeesPy documentation.
01143             Defaults to "Lobatto".
01144         @param max_iter (int, optional): Maximal number of iteration to reach the integretion
      convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01145         @param tol (float, optional): Tolerance for the integration convergence. Defaults to
      TOL_INTEGRATION (Units).
01146         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
      IDConvention to define it.
01147         """
01148         self.section = deepcopy(fiber.section)
01149         super().__init__(iNode_ID, jNode_ID, fiber.ID, geo_transf_ID,
```

```
01150             new_integration_ID=new_integration_ID, Ip=Ip, integration_type=integration_type,
      max_iter=max_iter, tol=tol, ele_ID= ele_ID)
01151         self.section_name_tag = self.section.name_tag
01152         self.UpdateStoredData()
01153         # Check length
01154         self._CheckL()
01155
01156
```

7.19.3 Member Data Documentation

7.19.3.1 section

`section`

Definition at line 1148 of file MemberModel.py.

7.19.3.2 section_name_tag

`section_name_tag`

Definition at line 1151 of file MemberModel.py.

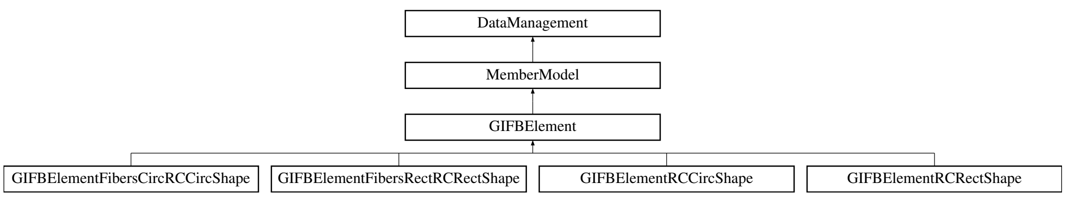
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.20 GIFBElement Class Reference

Class that handles the storage and manipulation of a Gradient-Inelastic Flexibility-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Inheritance diagram for GIFBElement:



Public Member Functions

- def __init__ (self, int iNode_ID, int jNode_ID, int fiber_ID, D_bars, fy, int geo_transf_ID, lambda_i=-1, lambda_j=-1, Lp=-1, Ip=-1, new_integration_ID=-1, min_tol=TOL_INTEGRATION, max_tol=TOL_↩ INTEGRATION ∗1e4, max_iter=MAX_ITER_INTEGRATION, ele_ID=-1)

    Constructor of the class.
- def ComputeIp (self)

    Compute the number of integration points with equal distance along the element.
- def ComputeLp (self)

    Method that computes the plastic length using Paulay 1992.
- def CreateMember (self)

    Method that initialises the member by calling the OpenSeesPy commands through various functions.
- def Record (self, str name_txt, str data_dir, force_rec=True, def_rec=True, time_rec=True)

    Implementation of the homonym abstract method.
- def RecordNodeDef (self, str name_txt, str data_dir, time_rec=True)

    Implementation of the homonym abstract method.
- def ReInit (self, lambda_i=-1, lambda_j=-1, Lp=-1, Ip=5, new_integration_ID=-1, ele_ID=-1)

    Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False)

    Implementation of the homonym abstract method.
- def UpdateStoredData (self)

    Implementation of the homonym abstract method.

Public Attributes

- D_bars
- data
- element_array
- element_ID
- fiber_ID
- fy
- geo_transf_ID
- Initialized
- iNode_ID
- Ip
- jNode_ID
- L
- lambda_i
- lambda_j
- Lp
- max_iter
- max_tol
- min_tol
- new_integration_ID
- section_name_tag

7.20.1 Detailed Description

Class that handles the storage and manipulation of a Gradient-Inelastic Flexibility-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

The integration technique is Simpson. For more information, see Sideris and Salehi 2016, 2017 and 2020.

Parameters

MemberModel	Parent abstract class.

Definition at line 1221 of file MemberModel.py.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 __init__()

```
def __init__ (
                 self,
                int iNode_ID,
                int jNode_ID,
                int fiber_ID,
                 D_bars,
                 fy,
                int geo_transf_ID,
                 lambda_i = -1,
                 lambda_j = -1,
                 Lp = -1,
                 Ip = -1,
                 new_integration_ID = -1,
                 min_tol = TOL_INTEGRATION,
                 max_tol = TOL_INTEGRATION*1e4,
                 max_iter = MAX_ITER_INTEGRATION,
                 ele_ID = -1 )
```

Constructor of the class.

Parameters

iNode_ID	(int): ID of the first end node.
jNode_ID	(int): ID of the second end node.
fiber_ID	(int): ID of the fiber section.
D_bars	(float): Diameter of the vertical reinforcing bars.
fy	(float): Yield stress of the reinforcing bars.
geo_transf_ID	(int): The geometric transformation (for more information, see OpenSeesPy documentation).
lambda_i	(float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end i.
lambda_j	(float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end j.
Lp	(float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
Ip	(int, optional): Number of integration points (min. 3). Defaults to 5.
new_integration_ID	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
min_tol	(float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).

Parameters

max_tol	(float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
max_iter	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	ID needs to be a positive integer.
NegativeValue	ID needs to be a positive integer.
NegativeValue	D_bars needs to be positive.
NegativeValue	fy needs to be positive.
NegativeValue	ID needs to be a positive integer.
NegativeValue	lambda_i needs to be positive.
NegativeValue	lambda_j needs to be positive.
NegativeValue	No plastic length defined.
NegativeValue	Lp needs to be positive, if different from -1.
NegativeValue	Ip needs to be a positive integer bigger than 3, if different from -1.
NegativeValue	ID needs to be a positive integer.
NegativeValue	min_tol needs to be positive.
NegativeValue	max_tol needs to be positive.
NegativeValue	max_iter needs to be a positive integer.
NegativeValue	ID needs to be a positive integer, if different from -1.

Reimplemented in GIFBElementFibersCircRCCircShape, GIFBElementFibersRectRCRectShape, GIFBElementRCCircShape, and GIFBElementRCRectShape.

Definition at line 1229 of file MemberModel.py.

```
01231         min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION, ele_ID = -1):
01232         """
01233         Constructor of the class.
01234 
01235         @param iNode_ID (int): ID of the first end node.
01236         @param jNode_ID (int): ID of the second end node.
01237         @param fiber_ID (int): ID of the fiber section.
01238         @param D_bars (float): Diameter of the vertical reinforcing bars.
01239         @param fy (float): Yield stress of the reinforcing bars.
01240         @param geo_transf_ID (int): The geometric transformation (for more information, see OpenSeesPy documentation).
01241         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge).
01242             Defaults to -1, e.g. plastic hinge in the end i.
01243         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge).
01244             Defaults to -1, e.g. plastic hinge in the end j.
01245         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
01246         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01247         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
01248         @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
01249         @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
01250         @param max_iter (int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01251         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
01252 
```

```
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: D_bars needs to be positive.
        @exception NegativeValue: fy needs to be positive.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: lambda_i needs to be positive.
        @exception NegativeValue: lambda_j needs to be positive.
        @exception NegativeValue: No plastic length defined.
        @exception NegativeValue: Lp needs to be positive, if different from -1.
        @exception NegativeValue: Ip needs to be a positive integer bigger than 3, if different from -1.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: min_tol needs to be positive.
        @exception NegativeValue: max_tol needs to be positive.
        @exception NegativeValue: max_iter needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer, if different from -1.
        """
        # Check
        if iNode_ID < 1: raise NegativeValue()
        if jNode_ID < 1: raise NegativeValue()
        if fiber_ID < 1: raise NegativeValue()
        if D_bars < 0: raise NegativeValue()
        if fy < 0: raise NegativeValue()
        if geo_transf_ID < 1: raise NegativeValue()
        if lambda_i != -1 and lambda_i < 0: raise NegativeValue()
        if lambda_j != -1 and lambda_j < 0: raise NegativeValue()
        if lambda_i == 0 and lambda_j == 0: print("!!!!!!! WARNING !!!!!!! No plastic length defined for element ID = {}".format(IDConvention(iNode_ID, jNode_ID)))
        if Lp != -1 and Lp < 0: raise NegativeValue()
        if Ip != -1 and Ip < 3: raise NegativeValue()
        if new_integration_ID != -1 and new_integration_ID < 1: raise NegativeValue()
        if min_tol < 0: raise NegativeValue()
        if max_tol < 0: raise NegativeValue()
        if max_iter < 0: raise NegativeValue()
        if ele_ID != -1 and ele_ID < 0: raise NegativeValue()

        # Arguments
        self.iNode_ID = iNode_ID
        self.jNode_ID = jNode_ID
        self.D_bars = D_bars
        self.fy = fy
        self.geo_transf_ID = geo_transf_ID
        self.fiber_ID = fiber_ID
        self.min_tol = min_tol
        self.max_tol = max_tol
        self.max_iter = max_iter

        # Initialized the parameters that are dependent from others
        self.section_name_tag = "None"
        self.Initialized = False
        self.ReInit(lambda_i, lambda_j, Lp, Ip, new_integration_ID, ele_ID)

```

7.20.3 Member Function Documentation

7.20.3.1 ComputeIp()

```
def ComputeIp (
            self )
```

Compute the number of integration points with equal distance along the element.

For more information, see Salehi and Sideris 2020.

Returns

int: Number of integration points

Definition at line 1439 of file MemberModel.py.

```
01439     def ComputeIp(self):
01440         """
01441         Compute the number of integration points with equal distance along the element. For more
      information, see Salehi and Sideris 2020.
01442 
01443         @returns int: Number of integration points
01444         """
01445         tmp = math.ceil(1.5*self.L/self.Lp + 1)
01446         if (tmp % 2) == 0:
01447             return tmp + 1
01448         else:
01449             return tmp
01450 
01451 
```

7.20.3.2 ComputeLp()

```
def ComputeLp (
                self )
```

Method that computes the plastic length using Paulay 1992.

Returns

double: Plastic length

Definition at line 1430 of file MemberModel.py.

```
01430     def ComputeLp(self):
01431         """
01432         Method that computes the plastic length using Paulay 1992.
01433 
01434         @returns double: Plastic length
01435         """
01436         return (0.08*self.L/m_unit + 0.022*self.D_bars/m_unit*self.fy/MPa_unit)*m_unit
01437 
01438 
```

7.20.3.3 CreateMember()

```
def CreateMember (
                self )
```

Method that initialises the member by calling the OpenSeesPy commands through various functions.

Definition at line 1396 of file MemberModel.py.

```
01396     def CreateMember(self):
01397         """
01398         Method that initialises the member by calling the OpenSeesPy commands through various
      functions.
01399         """
01400         self.element_array = [[self.element_ID, self.iNode_ID, self.jNode_ID]]
01401 
01402         # Define integration type
01403         beamIntegration('Simpson', self.new_integration_ID, self.fiber_ID, self.Ip)
01404 
01405         # Define element TODO: Dr. Salehi: lambda useless
01406         element('gradientInelasticBeamColumn', self.element_ID, self.iNode_ID, self.jNode_ID,
      self.geo_transf_ID,
01407             self.new_integration_ID, self.lambda_i, self.lambda_j, self.Lp, '-iter', self.max_iter,
      self.min_tol, self.max_tol)
01408 
01409         # Update class
01410         self.Initialized = True
01411         self.UpdateStoredData()
01412 
01413 
```

7.20.3.4 Record()

```
def Record (
        self,
        str name_txt,
        str data_dir,
        force_rec = True,
        def_rec = True,
        time_rec = True )
```

Implementation of the homonym abstract method.

See parent class MemberModel for detailed information.

Reimplemented from MemberModel.

Definition at line 1414 of file MemberModel.py.

```
01414     def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
01415         """
01416         Implementation of the homonym abstract method.
01417         See parent class MemberModel for detailed information.
01418         """
01419         super().Record(self.element_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec,
      time_rec=time_rec)
01420
01421
```

7.20.3.5 RecordNodeDef()

```
def RecordNodeDef (
        self,
        str name_txt,
        str data_dir,
        time_rec = True )
```

Implementation of the homonym abstract method.

See parent class MemberModel for detailed information.

Reimplemented from MemberModel.

Definition at line 1422 of file MemberModel.py.

```
01422     def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
01423         """
01424         Implementation of the homonym abstract method.
01425         See parent class MemberModel for detailed information.
01426         """
01427         super().RecordNodeDef(self.iNode_ID, self.jNode_ID, name_txt, data_dir, time_rec=time_rec)
01428
01429
```

7.20.3.6 ReInit()

```
def ReInit (
        self,
        lambda_i = -1,
        lambda_j = -1,
        Lp = -1,
        Ip = 5,
        new_integration_ID = -1,
        ele_ID = -1 )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

lambda_i	(float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end i.
lambda_j	(float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end j.
Lp	(float, optional): Plastic hinge length. Defaults to -1, e.g. computed here.
Ip	(int, optional): Number of integration points (min. 3). Defaults to 5.
new_integration_ID	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Definition at line 1304 of file MemberModel.py.

```
01304     def ReInit(self, lambda_i = -1, lambda_j = -1, Lp = -1, Ip = 5, new_integration_ID = -1, ele_ID = -1):
01305         """
01306         Implementation of the homonym abstract method.
01307         See parent class DataManagement for detailed information.
01308 
01309         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge).
01310             Defaults to -1, e.g. plastic hinge in the end i.
01311         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge).
01312             Defaults to -1, e.g. plastic hinge in the end j.
01313         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed here.
01314         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01315         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
01316         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
01317         """
01318         # Precompute some members
01319         iNode = np.array(nodeCoord(self.iNode_ID))
01320         jNode = np.array(nodeCoord(self.jNode_ID))
01321         self.L = np.linalg.norm(iNode-jNode)
01322         self.element_ID = IDConvention(self.iNode_ID, self.jNode_ID) if ele_ID == -1 else ele_ID
01323 
01324         # Arguments
01325         self.Lp = self.ComputeLp() if Lp == -1 else Lp
01326         self.Ip = self.ComputeIp() if Ip == -1 else Ip
01327         self.lambda_i = self.Lp/self.L if lambda_i == -1 else lambda_i
01328         self.lambda_j = self.Lp/self.L if lambda_j == -1 else lambda_j
01329         self.new_integration_ID = self.element_ID if new_integration_ID == -1 else new_integration_ID
01330 
01331         # Members
01332         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + " (modified)"
01333 
01334         # Data storage for loading/saving
01335         self.UpdateStoredData()
01336 
01337 
```

7.20.3.7 ShowInfo()

```
def ShowInfo (
        self,
        plot = False,
        block = False )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the material model. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 1365 of file MemberModel.py.

```
01365     def ShowInfo(self, plot = False, block = False):
01366         """
01367         Implementation of the homonym abstract method.
01368         See parent class DataManagement for detailed information.
01369 
01370         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
      False.
01371         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
      of the program everytime that a plot should pop up). Defaults to False.
01372         """
01373         print("")
01374         print("Requested info for GIFBElement member model, ID = {}".format(self.element_ID))
01375         print("Fiber associated, ID = {} ".format(self.fiber_ID))
01376         print("Integration type 'Simpson', ID = {}".format(self.new_integration_ID))
01377         print("Section associated {} ".format(self.section_name_tag))
01378         print("Length L = {} m".format(self.L/m_unit))
01379         print("Diameter of the reinforcing bars D_bars = {} mm2".format(self.D_bars/mm2_unit))
01380         print("Reinforcing bar steel strength fy = {} MPa".format(self.fy/MPa_unit))
01381         print("Plastic length Lp = {} mm".format(self.Lp/mm_unit))
01382         print("Number of integration points along the element Ip = {}, max iter = {}, (min, max tol) =
      ({},{})".format(self.Ip, self.max_iter, self.min_tol, self.max_tol))
01383         print("Lambda_i = {} and lambda_j = {}".format(self.lambda_i, self.lambda_j))
01384         print("Geometric transformation = {}".format(self.geo_transf_ID))
01385         print("")
01386 
01387         if plot:
01388             if self.Initialized:
01389                 plot_member(self.element_array, "GIFB Element, ID = {}".format(self.element_ID))
01390                 if block:
01391                     plt.show()
01392             else:
01393                 print("The GIFBElement is not initialized (element not created), ID =
      {}".format(self.element_ID))
01394 
01395 
```

7.20.3.8 UpdateStoredData()

```
def UpdateStoredData (
                self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 1339 of file MemberModel.py.

```
01339     def UpdateStoredData(self):
01340         """
01341         Implementation of the homonym abstract method.
01342         See parent class DataManagement for detailed information.
01343         """
01344         self.data = [["INFO_TYPE", "GIFBElement"], # Tag for differentiating different data
01345             ["element_ID", self.element_ID],
01346             ["section_name_tag", self.section_name_tag],
01347             ["L", self.L],
01348             ["D_bars", self.D_bars],
01349             ["fy", self.fy],
01350             ["Lp", self.Lp],
01351             ["Ip", self.Ip],
01352             ["iNode_ID", self.iNode_ID],
01353             ["lambda_i", self.lambda_i],
01354             ["jNode_ID", self.jNode_ID],
01355             ["lambda_j", self.lambda_j],
01356             ["fiber_ID", self.fiber_ID],
```

```
            ["new_integration_ID", self.new_integration_ID],
            ["min_tol", self.min_tol],
            ["max_tol", self.max_tol],
            ["max_iter", self.max_iter],
            ["tranf_ID", self.geo_transf_ID],
            ["Initialized", self.Initialized]]
```

7.20.4 Member Data Documentation

7.20.4.1 D_bars

`D_bars`

Definition at line 1291 of file MemberModel.py.

7.20.4.2 data

`data`

Definition at line 1344 of file MemberModel.py.

7.20.4.3 element_array

`element_array`

Definition at line 1400 of file MemberModel.py.

7.20.4.4 element_ID

`element_ID`

Definition at line 1322 of file MemberModel.py.

7.20.4.5 fiber_ID

`fiber_ID`

Definition at line 1294 of file MemberModel.py.

7.20.4.6 fy

```
fy
```

Definition at line 1292 of file MemberModel.py.

7.20.4.7 geo_transf_ID

```
geo_transf_ID
```

Definition at line 1293 of file MemberModel.py.

7.20.4.8 Initialized

```
Initialized
```

Definition at line 1301 of file MemberModel.py.

7.20.4.9 iNode_ID

```
iNode_ID
```

Definition at line 1289 of file MemberModel.py.

7.20.4.10 Ip

```
Ip
```

Definition at line 1326 of file MemberModel.py.

7.20.4.11 jNode_ID

```
jNode_ID
```

Definition at line 1290 of file MemberModel.py.

7.20.4.12 L

```
L
```

Definition at line 1321 of file MemberModel.py.

7.20.4.13 lambda_i

```
lambda_i
```

Definition at line 1327 of file MemberModel.py.

7.20.4.14 lambda_j

```
lambda_j
```

Definition at line 1328 of file MemberModel.py.

7.20.4.15 Lp

```
Lp
```

Definition at line 1325 of file MemberModel.py.

7.20.4.16 max_iter

```
max_iter
```

Definition at line 1297 of file MemberModel.py.

7.20.4.17 max_tol

```
max_tol
```

Definition at line 1296 of file MemberModel.py.

7.20.4.18 min_tol

`min_tol`

Definition at line 1295 of file MemberModel.py.

7.20.4.19 new_integration_ID

`new_integration_ID`

Definition at line 1329 of file MemberModel.py.

7.20.4.20 section_name_tag

`section_name_tag`

Definition at line 1300 of file MemberModel.py.

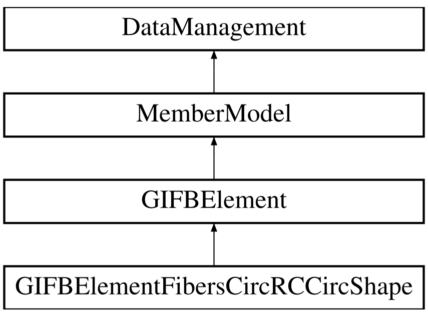
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.21 GIFBElementFibersCircRCCircShape Class Reference

Class that is the children of GIFBElement and combine the class FibersCircRCCircShape (fiber section) to retrieve the information needed.

Inheritance diagram for GIFBElementFibersCircRCCircShape:



Public Member Functions

- def __init__ (self, int iNode_ID, int jNode_ID, FibersCircRCCircShape fib, int geo_transf_ID, lambda_i=-1, lambda_j=-1, Lp=-1, Ip=-1, new_integration_ID=-1, min_tol=TOL_INTEGRATION, max_tol=TOL_INTEGRATION ∗1e4, max_iter=MAX_ITER_INTEGRATION, ele_ID=-1)

  Constructor of the class.

Public Attributes

- section
- section_name_tag

7.21.1 Detailed Description

Class that is the children of GIFBElement and combine the class FibersCircRCCircShape (fiber section) to retrieve the information needed.

Parameters

GIFBElement	Parent class.

Definition at line 1568 of file MemberModel.py.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 __init__()

```
def __init__ (
                self,
               int iNode_ID,
               int jNode_ID,
               FibersCircRCCircShape fib,
               int geo_transf_ID,
                lambda_i = -1,
                lambda_j = -1,
                Lp = -1,
                Ip = -1,
                new_integration_ID = -1,
                min_tol = TOL_INTEGRATION,
                max_tol = TOL_INTEGRATION*1e4,
                max_iter = MAX_ITER_INTEGRATION,
                ele_ID = -1 )
```

Constructor of the class.

Parameters

iNode_ID	(int): ID of the first end node.
jNode_ID	(int): ID of the second end node.
fib	(FibersCircRCCircShape): FibersCircRCCircShape fiber section object.
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).

Parameters

lambda_i	(float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end i.
lambda_j	(float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end j.
Lp	(float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
Ip	(int, optional): Number of integration points (min. 3). Defaults to 5.
new_integration_ID	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
min_tol	(float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
max_tol	(float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION∗1e4.
max_iter	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from GIFBElement.

Definition at line 1574 of file MemberModel.py.

```
01576         min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION, ele_ID = -1):
01577         """
01578         Constructor of the class.
01579 
01580         @param iNode_ID (int): ID of the first end node.
01581         @param jNode_ID (int): ID of the second end node.
01582         @param fib (FibersCircRCCircShape): FibersCircRCCircShape fiber section object.
01583         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
01584         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge).
01585             Defaults to -1, e.g. plastic hinge in the end i.
01586         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge).
01587             Defaults to -1, e.g. plastic hinge in the end j.
01588         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
01589         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01590         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
01591         @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
01592         @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
01593         @param max_iter (int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01594         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
01595         """
01596         self.section = deepcopy(fib.section)
01597         super().__init__(iNode_ID, jNode_ID, fib.ID, self.section.D_bars, self.section.fy, geo_transf_ID,
01598             lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
01599             min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
01600         self.section_name_tag = self.section.name_tag
01601         self.UpdateStoredData()
01602         # Check length
01603         self._CheckL()
01604 
01605 
01606 
01607 
```

7.21.3 Member Data Documentation

7.21.3.1 section

```
section
```

Definition at line 1596 of file MemberModel.py.

7.21.3.2 section_name_tag

```
section_name_tag
```

Definition at line 1600 of file MemberModel.py.

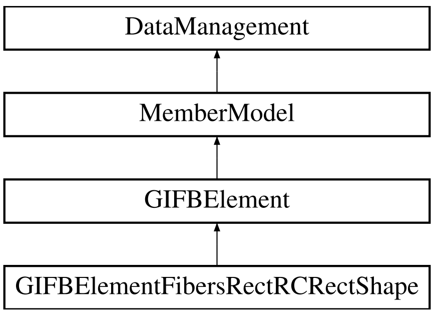
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.22 GIFBElementFibersRectRCRectShape Class Reference

Class that is the children of GIFBElement and combine the class FibersRectRCRectShape (fiber section) to retrieve the information needed.

Inheritance diagram for GIFBElementFibersRectRCRectShape:



Public Member Functions

- def __init__ (self, int iNode_ID, int jNode_ID, FibersRectRCRectShape fib, int geo_transf_ID, lambda_i=-1, lambda_j=-1, Lp=-1, Ip=-1, new_integration_ID=-1, min_tol=TOL_INTEGRATION, max_tol=TOL_↩ INTEGRATION ∗1e4, max_iter=MAX_ITER_INTEGRATION, ele_ID=-1)

    Constructor of the class.

Public Attributes

- section
- section_name_tag

7.22.1 Detailed Description

Class that is the children of GIFBElement and combine the class FibersRectRCRectShape (fiber section) to retrieve the information needed.

Parameters

GIFBElement	Parent class.

Definition at line 1491 of file MemberModel.py.

7.22.2 Constructor & Destructor Documentation

7.22.2.1 __init__()

```
def __init__ (
                 self,
                int iNode_ID,
                int jNode_ID,
                FibersRectRCRectShape fib,
                int geo_transf_ID,
                 lambda_i = -1,
                 lambda_j = -1,
                 Lp = -1,
                 Ip = -1,
                 new_integration_ID = -1,
                 min_tol = TOL_INTEGRATION,
                 max_tol = TOL_INTEGRATION*1e4,
                 max_iter = MAX_ITER_INTEGRATION,
                 ele_ID = -1 )
```

Constructor of the class.

Parameters

iNode_ID	(int): ID of the first end node.
jNode_ID	(int): ID of the second end node.
fib	(FibersRectRCRectShape): FibersRectRCRectShape fiber section object.
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
lambda_i	(float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end i.
lambda_j	(float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end j.
Lp	(float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
Ip	(int, optional): Number of integration points (min. 3). Defaults to 5.
new_integration_ID	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
min_tol	(float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
max_tol	(float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION∗1e4.
max_iter	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from GIFBElement.

Definition at line 1497 of file MemberModel.py.

```
01499         min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION,
      ele_ID = -1):
01500         """
01501         Constructor of the class.
01502 
01503         @param iNode_ID (int): ID of the first end node.
01504         @param jNode_ID (int): ID of the second end node.
01505         @param fib (FibersRectRCRectShape): FibersRectRCRectShape fiber section object.
01506         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
      documentation).
01507         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
      end i (0 = no plastic hinge).
01508             Defaults to -1, e.g. plastic hinge in the end i.
01509         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
      end j (0 = no plastic hinge).
01510             Defaults to -1, e.g. plastic hinge in the end j.
01511         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
01512         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01513         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
      e.g. computed in ReInit().
01514         @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults
      to TOL_INTEGRATION (Units).
01515         @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults
      to TOL_INTEGRATION*1e4.
01516         @param max_iter (int, optional): Maximal number of iteration to reach the integretion
      convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01517         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
      IDConvention to define it.
01518         """
01519         self.section = deepcopy(fib.section)
01520         super().__init__(iNode_ID, jNode_ID, fib.ID, self.section.D_bars, self.section.fy,
      geo_transf_ID,
01521             lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
01522             min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
01523         self.section_name_tag = self.section.name_tag
01524         self.UpdateStoredData()
01525         # Check length
01526         self._CheckL()
01527 
01528 
```

7.22.3 Member Data Documentation

7.22.3.1 section

section

Definition at line 1519 of file MemberModel.py.

7.22.3.2 section_name_tag

section_name_tag

Definition at line 1523 of file MemberModel.py.

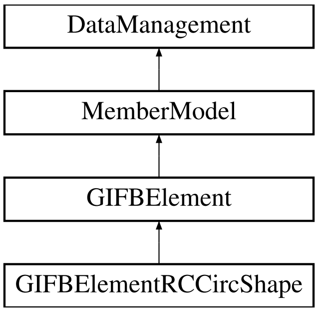
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.23 GIFBElementRCCircShape Class Reference

Class that is the children of GIFBElement and combine the class RCCircShape (section) to retrieve the information needed.

Inheritance diagram for GIFBElementRCCircShape:



Public Member Functions

- def __init__ (self, int iNode_ID, int jNode_ID, int fiber_ID, RCCircShape section, int geo_transf_ID, lambda_i=-1, lambda_j=-1, Lp=-1, Ip=-1, new_integration_ID=-1, min_tol=TOL_INTEGRATION, max_tol=TOL↵_INTEGRATION ∗1e4, max_iter=MAX_ITER_INTEGRATION, ele_ID=-1)

    Constructor of the class.

Public Attributes

- section
- section_name_tag

7.23.1 Detailed Description

Class that is the children of GIFBElement and combine the class RCCircShape (section) to retrieve the information needed.

Parameters

GIFBElement	Parent class.

Definition at line 1529 of file MemberModel.py.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 __init__()

```
def __init__ (
             self,
            int iNode_ID,
            int jNode_ID,
            int fiber_ID,
            RCCircShape section,
            int geo_transf_ID,
             lambda_i = -1,
             lambda_j = -1,
             Lp = -1,
             Ip = -1,
             new_integration_ID = -1,
             min_tol = TOL_INTEGRATION,
             max_tol = TOL_INTEGRATION*1e4,
             max_iter = MAX_ITER_INTEGRATION,
             ele_ID = -1 )
```

Constructor of the class.

Parameters

iNode_ID	(int): ID of the first end node.
jNode_ID	(int): ID of the second end node.
fiber_ID	(int): ID of the fiber section.
section	(RCCircShape): RCCircShape section object.
geo_transf_ID	(int): The geometric transformation (for more information, see OpenSeesPy documentation).
lambda_i	(float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end i.
lambda_j	(float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end j.
Lp	(float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
Ip	(int, optional): Number of integration points (min. 3). Defaults to 5.
new_integration_ID	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
min_tol	(float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
max_tol	(float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
max_iter	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from GIFBElement.

Definition at line 1535 of file MemberModel.py.

```
        min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION,
    ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param fiber_ID (int): ID of the fiber section.
        @param section (RCCircShape): RCCircShape section object.
```

```
01545        @param geo_transf_ID (int): The geometric transformation (for more information, see OpenSeesPy
    documentation).
01546        @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
    end i (0 = no plastic hinge).
01547            Defaults to -1, e.g. plastic hinge in the end i.
01548        @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
    end j (0 = no plastic hinge).
01549            Defaults to -1, e.g. plastic hinge in the end j.
01550        @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
01551        @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01552        @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
    e.g. computed in ReInit().
01553        @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults
    to TOL_INTEGRATION (Units).
01554        @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults
    to TOL_INTEGRATION*1e4.
01555        @param max_iter (int, optional): Maximal number of iteration to reach the integretion
    convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01556        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
    IDConvention to define it.
01557        """
01558        self.section = deepcopy(section)
01559        super().__init__(iNode_ID, jNode_ID, fiber_ID, section.D_bars, section.fy, geo_transf_ID,
01560            lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
01561            min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
01562        self.section_name_tag = section.name_tag
01563        self.UpdateStoredData()
01564        # Check length
01565        self._CheckL()
01566
01567
```

7.23.3 Member Data Documentation

7.23.3.1 section

`section`

Definition at line 1558 of file MemberModel.py.

7.23.3.2 section_name_tag

`section_name_tag`

Definition at line 1562 of file MemberModel.py.

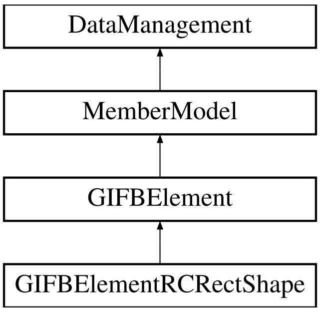
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.24 GIFBElementRCRectShape Class Reference

Class that is the children of GIFBElement and combine the class RCRectShape (section) to retrieve the information needed.

Inheritance diagram for GIFBElementRCRectShape:



Public Member Functions

- def __init__ (self, int iNode_ID, int jNode_ID, int fiber_ID, RCRectShape section, int geo_transf_ID, lambda_i=-1, lambda_j=-1, Lp=-1, Ip=-1, new_integration_ID=-1, min_tol=TOL_INTEGRATION, max_tol=TOL↩_INTEGRATION ∗1e4, max_iter=MAX_ITER_INTEGRATION, ele_ID=-1)

    Constructor of the class.

Public Attributes

- section
- section_name_tag

7.24.1 Detailed Description

Class that is the children of GIFBElement and combine the class RCRectShape (section) to retrieve the information needed.

Parameters

GIFBElement	Parent class.

Definition at line 1452 of file MemberModel.py.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 __init__()

```
def __init__ (
             self,
            int iNode_ID,
            int jNode_ID,
            int fiber_ID,
            RCRectShape section,
            int geo_transf_ID,
             lambda_i = -1,
             lambda_j = -1,
             Lp = -1,
             Ip = -1,
             new_integration_ID = -1,
             min_tol = TOL_INTEGRATION,
             max_tol = TOL_INTEGRATION*1e4,
             max_iter = MAX_ITER_INTEGRATION,
             ele_ID = -1 )
```

Constructor of the class.

Parameters

iNode_ID	(int): ID of the first end node.
jNode_ID	(int): ID of the second end node.
fiber_ID	(int): ID of the fiber section.
section	(RCRectShape): RCRectShape section object.
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
lambda_i	(float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end i.
lambda_j	(float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end j.
Lp	(float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
Ip	(int, optional): Number of integration points (min. 3). Defaults to 5.
new_integration_ID	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
min_tol	(float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
max_tol	(float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
max_iter	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from GIFBElement.

Definition at line 1458 of file MemberModel.py.

```
01460         min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION,
        ele_ID = -1):
01461         """
01462         Constructor of the class.
01463 
01464         @param iNode_ID (int): ID of the first end node.
01465         @param jNode_ID (int): ID of the second end node.
01466         @param fiber_ID (int): ID of the fiber section.
01467         @param section (RCRectShape): RCRectShape section object.
```

```
01468         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
      documentation).
01469         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
      end i (0 = no plastic hinge).
01470             Defaults to -1, e.g. plastic hinge in the end i.
01471         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
      end j (0 = no plastic hinge).
01472             Defaults to -1, e.g. plastic hinge in the end j.
01473         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
01474         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01475         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
      e.g. computed in ReInit().
01476         @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults
      to TOL_INTEGRATION (Units).
01477         @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults
      to TOL_INTEGRATION*1e4.
01478         @param max_iter (int, optional): Maximal number of iteration to reach the integretion
      convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01479         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
      IDConvention to define it.
01480         """
01481         self.section = deepcopy(section)
01482         super().__init__(iNode_ID, jNode_ID, fiber_ID, section.D_bars, section.fy, geo_transf_ID,
01483             lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
01484             min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
01485         self.section_name_tag = section.name_tag
01486         self.UpdateStoredData()
01487         # Check length
01488         self._CheckL()
01489
01490
```

7.24.3 Member Data Documentation

7.24.3.1 section

```
section
```

Definition at line 1481 of file MemberModel.py.

7.24.3.2 section_name_tag

```
section_name_tag
```

Definition at line 1485 of file MemberModel.py.

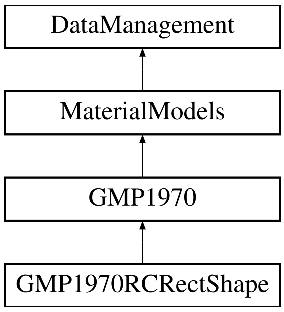
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.25 GMP1970 Class Reference

Class that stores funcions and material properties of the vertical steel reinforcement bars with Giuffré, Menegotto and Pinto 1970 as the material model and the OpenSeesPy command type used to model it is Steel02.

Inheritance diagram for GMP1970:



Public Member Functions

- def __init__ (self, int ID, fy, Ey, b=0.02, R0=20, cR1=0.9, cR2=0.08, a1=0.039, a2=1.0, a3=0.029, a4=1.0)

  Constructor of the class.
- def CheckApplicability (self)

  Implementation of the homonym abstract method.
- def ReInit (self)

  Implementation of the homonym abstract method.
- def ShowInfo (self)

  Implementation of the homonym abstract method.
- def Steel02 (self)

  Generate the material model Steel02 uniaxial Giuffre-Menegotto-Pinto steel material with isotropic strain hardening.
- def UpdateStoredData (self)

  Implementation of the homonym abstract method.

Public Attributes

- a1
- a2
- a3
- a4
- b
- cR1
- cR2
- data
- Ey
- fy
- ID
- Initialized
- R0
- section_name_tag

7.25.1 Detailed Description

Class that stores funcions and material properties of the vertical steel reinforcement bars with Giuffré, Menegotto and Pinto 1970 as the material model and the OpenSeesPy command type used to model it is Steel02.

For more information about the empirical model for the computation of the parameters, see Giuffré, Menegotto and Pinto 1970 and Carreno et Al. 2020.

Parameters

MaterialModels	Parent abstract class.

Definition at line 2470 of file MaterialModels.py.

7.25.2 Constructor & Destructor Documentation

7.25.2.1 __init__()

```
def __init__ (
              self,
             int ID,
              fy,
              Ey,
              b = 0.02,
              R0 = 20,
              cR1 = 0.9,
              cR2 = 0.08,
              a1 = 0.039,
              a2 = 1.0,
              a3 = 0.029,
              a4 = 1.0 )
```

Constructor of the class.

The parameters are suggested as exposed in Carreno et Al. 2020 but also the one suggested by OpenSeesPy documentation are reliable (b = 0.015, R0 = 10, cR1 = 0.925, cR2 = 0.15).

Parameters

ID	(int): Unique material model ID.
fy	(float): Steel yield strength.
Ey	(float): Young modulus.
b	(float, optional): Strain-hardening ratio. Defaults to 0.02, according to Carreno et Al. 2020.
R0	(int, optional): First parameter to control the transition from elastic to plastic branches. Defaults to 20, according to Carreno et Al. 2020.
cR1	(float, optional): Second parameter to control the transition from elastic to plastic branches. Defaults to 0.9, according to Carreno et Al. 2020.
cR2	(float, optional): Third parameter to control the transition from elastic to plastic branches. Defaults to 0.08, according to Carreno et Al. 2020.
a1	(float, optional): Isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain. Defaults to 0.039, according to Carreno et Al. 2020.
a2	(float, optional): Coupled with a1. Defaults to 1.0, according to Carreno et Al. 2020.
a3	(float, optional): Isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain. Defaults to 0.029, according to Carreno et Al. 2020.
a4	(float, optional): Coupled with a3. Defaults to 1.0, according to Carreno et Al. 2020.

Exceptions

NegativeValue	ID needs to be a positive integer.

Reimplemented in GMP1970RCRectShape.

Definition at line 2478 of file MaterialModels.py.

```
02478     def __init__(self, ID: int, fy, Ey, b = 0.02, R0 = 20, cR1 = 0.9, cR2 = 0.08, a1 = 0.039, a2 = 1.0, a3 = 0.029, a4 = 1.0):
02479         """
02480         Constructor of the class. The parameters are suggested as exposed in Carreno et Al. 2020 but also the one suggested by OpenSeesPy documentation are reliable
02481             (b = 0.015, R0 = 10, cR1 = 0.925, cR2 = 0.15).
02482 
02483         @param ID (int): Unique material model ID.
02484         @param fy (float): Steel yield strength.
02485         @param Ey (float): Young modulus.
02486         @param b (float, optional): Strain-hardening ratio. Defaults to 0.02, according to Carreno et Al. 2020.
02487         @param R0 (int, optional): First parameter to control the transition from elastic to plastic branches. Defaults to 20, according to Carreno et Al. 2020.
02488         @param cR1 (float, optional): Second parameter to control the transition from elastic to plastic branches. Defaults to 0.9, according to Carreno et Al. 2020.
02489         @param cR2 (float, optional): Third parameter to control the transition from elastic to plastic branches. Defaults to 0.08, according to Carreno et Al. 2020.
02490         @param a1 (float, optional): Isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain.
02491             Defaults to 0.039, according to Carreno et Al. 2020.
02492         @param a2 (float, optional): Coupled with a1. Defaults to 1.0, according to Carreno et Al. 2020.
02493         @param a3 (float, optional): Isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain.
02494             Defaults to 0.029, according to Carreno et Al. 2020.
02495         @param a4 (float, optional): Coupled with a3. Defaults to 1.0, according to Carreno et Al. 2020.
02496 
02497         @exception NegativeValue: ID needs to be a positive integer.
02498         """
02499         # Check
02500         if ID < 1: raise NegativeValue()
02501 
02502         # Arguments
02503         self.ID = ID
02504         self.fy = fy
02505         self.Ey = Ey
02506         self.b = b
02507         self.R0 = R0
02508         self.cR1 = cR1
02509         self.cR2 = cR2
02510         self.a1 = a1
02511         self.a2 = a2
02512         self.a3 = a3
02513         self.a4 = a4
02514 
02515         # Initialized the parameters that are dependent from others
02516         self.section_name_tag = "None"
02517         self.Initialized = False
02518         self.ReInit()
02519 
```

7.25.3 Member Function Documentation

7.25.3.1 CheckApplicability()

```
def CheckApplicability (
            self )
```

Implementation of the homonym abstract method.

See parent class MaterialModels for detailed information.

Reimplemented from MaterialModels.

Definition at line 2575 of file MaterialModels.py.

```
    def CheckApplicability(self):
        """
        Implementation of the homonym abstract method.
        See parent class MaterialModels for detailed information.
        """
        Check = True
        # No checks
        if not Check:
            print("The validity of the equations is not fullfilled.")
            print("!!!!!!! WARNING !!!!!!! Check material model of GMP1970, ID=", self.ID)
            print("")


```

7.25.3.2 ReInit()

```
def ReInit (
            self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 2520 of file MaterialModels.py.

```
    def ReInit(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        # Check applicability
        self.CheckApplicability()

        # Members
        if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredData()


```

7.25.3.3 ShowInfo()

```
def ShowInfo (
            self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 2557 of file MaterialModels.py.

```
    def ShowInfo(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        print("")
        print("Requested info for GMP1970 (Giuffré-Menegotto-Pinto) material model Parameters, ID = {}".format(self.ID))
```

```
        print("Section associated: {} ".format(self.section_name_tag))
        print("Yield stress fy = {} MPa".format(self.fy/MPa_unit))
        print("Young modulus Ey = {} MPa".format(self.Ey/MPa_unit))
        print("Strain hardening ratio b = {}".format(self.b))
        print("Bauschinger effect factors R0 = {}, cR1 = {} and cR2 = {}".format(self.R0, self.cR1, self.cR2))
        print("Isotropic hardening factors a1 = {}, a2 = {}, a3 = {} and a4 = {}".format(self.a1, self.a2, self.a3, self.a4))
        print("")

        #TODO: add plot option (difficult to implement)


```

7.25.3.4 Steel02()

```
def Steel02 (
        self )
```

Generate the material model Steel02 uniaxial Giuffre-Menegotto-Pinto steel material with isotropic strain hardening.

See _Steel02 function for more information.

Definition at line 2588 of file MaterialModels.py.

```
02588     def Steel02(self):
02589         """
02590         Generate the material model Steel02 uniaxial Giuffre-Menegotto-Pinto steel material with isotropic strain hardening.
02591         See _Steel02 function for more information.
02592         """
02593         _Steel02(self.ID, self.fy, self.Ey, self.b, self.R0, self.cR1, self.cR2, self.a1, self.a2, self.a3, self.a4)
02594         self.Initialized = True
02595         self.UpdateStoredData()
02596
02597
```

7.25.3.5 UpdateStoredData()

```
def UpdateStoredData (
        self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 2536 of file MaterialModels.py.

```
02536     def UpdateStoredData(self):
02537         """
02538         Implementation of the homonym abstract method.
02539         See parent class DataManagement for detailed information.
02540         """
02541         self.data = [["INFO_TYPE", "GMP1970"], # Tag for differentiating different data
02542             ["ID", self.ID],
02543             ["section_name_tag", self.section_name_tag],
02544             ["fy", self.fy],
02545             ["Ey", self.Ey],
02546             ["b", self.b],
02547             ["R0", self.R0],
02548             ["cR1", self.cR1],
02549             ["cR2", self.cR2],
02550             ["a1", self.a1],
02551             ["a2", self.a2],
02552             ["a3", self.a3],
02553             ["a4", self.a4],
02554             ["Initialized", self.Initialized]]
02555
02556
```

7.25.4 Member Data Documentation

7.25.4.1 a1

```
a1
```

Definition at line 2510 of file MaterialModels.py.

7.25.4.2 a2

```
a2
```

Definition at line 2511 of file MaterialModels.py.

7.25.4.3 a3

```
a3
```

Definition at line 2512 of file MaterialModels.py.

7.25.4.4 a4

```
a4
```

Definition at line 2513 of file MaterialModels.py.

7.25.4.5 b

```
b
```

Definition at line 2506 of file MaterialModels.py.

7.25.4.6 cR1

```
cR1
```

Definition at line 2508 of file MaterialModels.py.

7.25.4.7 cR2

```
cR2
```

Definition at line 2509 of file MaterialModels.py.

7.25.4.8 data

```
data
```

Definition at line 2541 of file MaterialModels.py.

7.25.4.9 Ey

```
Ey
```

Definition at line 2505 of file MaterialModels.py.

7.25.4.10 fy

```
fy
```

Definition at line 2504 of file MaterialModels.py.

7.25.4.11 ID

```
ID
```

Definition at line 2503 of file MaterialModels.py.

7.25.4.12 Initialized

```
Initialized
```

Definition at line 2517 of file MaterialModels.py.

7.25.4.13 R0

```
R0
```

Definition at line 2507 of file MaterialModels.py.

7.25.4.14 section_name_tag

```
section_name_tag
```

Definition at line 2516 of file MaterialModels.py.

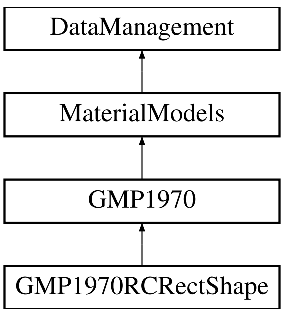
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.26 GMP1970RCRectShape Class Reference

Class that is the children of GMP1970 and combine the class RCRectShape (section) to retrieve the information needed.

Inheritance diagram for GMP1970RCRectShape:



Public Member Functions

- def __init__ (self, int ID, RCRectShape section, b=0.02, R0=20.0, cR1=0.9, cR2=0.08, a1=0.039, a2=1.0, a3=0.029, a4=1.0)

    Constructor of the class.

Public Attributes

- section
- section_name_tag

7.26.1 Detailed Description

Class that is the children of GMP1970 and combine the class RCRectShape (section) to retrieve the information needed.

Parameters

GMP1970	Parent class.

Definition at line 2598 of file MaterialModels.py.

7.26.2 Constructor & Destructor Documentation

7.26.2.1 __init__()

```
def __init__ (
        self,
        int ID,
        RCRectShape section,
        b = 0.02,
        R0 = 20.0,
        cR1 = 0.9,
        cR2 = 0.08,
        a1 = 0.039,
        a2 = 1.0,
        a3 = 0.029,
        a4 = 1.0 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class RCRectShape. The copy of the section passed is stored in the member variable self.section.

Parameters

ID	(int): Unique material model ID.
section	(RCRectShape): RCRectShape section object.
b	(float, optional): Strain-hardening ratio. Defaults to 0.02, according to Carreno et Al. 2020.
R0	(int, optional): First parameter to control the transition from elastic to plastic branches. Defaults to 20, according to Carreno et Al. 2020.

Parameters

cR1	(float, optional): Second parameter to control the transition from elastic to plastic branches. Defaults to 0.9, according to Carreno et Al. 2020.
cR2	(float, optional): Third parameter to control the transition from elastic to plastic branches. Defaults to 0.08, according to Carreno et Al. 2020.
a1	(float, optional): Isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain. Defaults to 0.039, according to Carreno et Al. 2020.
a2	(float, optional): Coupled with a1. Defaults to 1.0, according to Carreno et Al. 2020.
a3	(float, optional): Isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain. Defaults to 0.029, according to Carreno et Al. 2020.
a4	(float, optional): Coupled with a3. Defaults to 1.0, according to Carreno et Al. 2020.

Reimplemented from GMP1970.

Definition at line 2604 of file MaterialModels.py.

```
02604     def __init__(self, ID: int, section: RCRectShape, b=0.02, R0=20.0, cR1=0.9, cR2=0.08, a1=0.039, a2=1.0, a3=0.029, a4=1.0):
02605         """
02606         Constructor of the class. It passes the arguments into the parent class to generate the combination of the parent class
02607             and the section class RCRectShape.
02608         The copy of the section passed is stored in the member variable self.section.
02609 
02610         @param ID (int): Unique material model ID.
02611         @param section (RCRectShape): RCRectShape section object.
02612         @param b (float, optional): Strain-hardening ratio. Defaults to 0.02, according to Carreno et Al. 2020.
02613         @param R0 (int, optional): First parameter to control the transition from elastic to plastic branches. Defaults to 20, according to Carreno et Al. 2020.
02614         @param cR1 (float, optional): Second parameter to control the transition from elastic to plastic branches. Defaults to 0.9, according to Carreno et Al. 2020.
02615         @param cR2 (float, optional): Third parameter to control the transition from elastic to plastic branches. Defaults to 0.08, according to Carreno et Al. 2020.
02616         @param a1 (float, optional): Isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain.
02617             Defaults to 0.039, according to Carreno et Al. 2020.
02618         @param a2 (float, optional): Coupled with a1. Defaults to 1.0, according to Carreno et Al. 2020.
02619         @param a3 (float, optional): Isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain.
02620             Defaults to 0.029, according to Carreno et Al. 2020.
02621         @param a4 (float, optional): Coupled with a3. Defaults to 1.0, according to Carreno et Al. 2020.
02622         """
02623         self.section = deepcopy(section)
02624         super().__init__(ID, section.fy, section.Ey, b=b, R0=R0, cR1=cR1, cR2=cR2, a1=a1, a2=a2, a3=a3, a4=a4)
02625         self.section_name_tag = section.name_tag
02626         self.UpdateStoredData()
02627 
02628 
```

7.26.3 Member Data Documentation

7.26.3.1 section

`section`

Definition at line 2623 of file MaterialModels.py.

7.26.3.2 section_name_tag

`section_name_tag`

Definition at line 2625 of file MaterialModels.py.

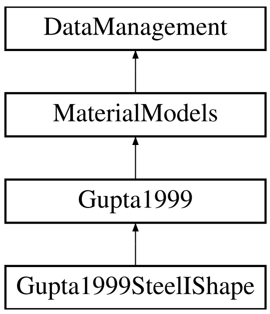
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.27 Gupta1999 Class Reference

Class that stores funcions and material properties of a steel double symmetric I-shape profile with Gupta 1999 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.

Inheritance diagram for Gupta1999:



Public Member Functions

- def __init__ (self, int ID, d_c, bf_c, tf_c, I_c, d_b, tf_b, Fy, E, t_p, t_dp=0.0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0.0, dmg2=0.0, beta=0.0, safety_factor=False)

    Constructor of the class.
- def CheckApplicability (self)

    Implementation of the homonym abstract method.
- def Hysteretic (self)

    Generate the material model Hysteretic (Gupta 1999) using the computed parameters.
- def ReInit (self)

    Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False)

    Implementation of the homonym abstract method.
- def UpdateStoredData (self)

    Implementation of the homonym abstract method.

Public Attributes

- a_s
- beam_section_name_tag
- beta
- bf_c
- col_section_name_tag
- d_b
- d_c
- data
- dmg1
- dmg2
- E
- Fy
- G
- gamma1_y
- gamma2_y
- gamma3_y
- I_c
- ID
- Initialized
- Ke
- Kp
- M1y
- M2y
- M3y
- pinchx
- pinchy
- Ry
- t_dp
- t_p
- t_pz
- tf_b
- tf_c
- Vy

7.27.1 Detailed Description

Class that stores funcions and material properties of a steel double symmetric I-shape profile with Gupta 1999 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.

The material model is valid only if the column is continuous. For more information about the empirical model for the computation of the parameters, see Gupta 1999.

Parameters

MaterialModels	Parent abstract class.

Definition at line 522 of file MaterialModels.py.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 __init__()

```
def __init__ (
             self,
            int ID,
             d_c,
             bf_c,
             tf_c,
             I_c,
             d_b,
             tf_b,
             Fy,
             E,
             t_p,
             t_dp = 0.0,
             a_s = 0.03,
             pinchx = 0.25,
             pinchy = 0.75,
             dmg1 = 0.0,
             dmg2 = 0.0,
             beta = 0.0,
             safety_factor = False )
```

Constructor of the class.

Parameters

ID	(int): Unique material model ID.
d_c	(float): Column depth.
bf_c	(float): Column flange width.
tf_c	(float): Column flange thickness.
I_c	(float): Column moment of inertia (strong axis).
d_b	(float): Beam depth.
tf_b	(float): Beam flange thickness.
Fy	(float): Yield strength (if assume continous column, Fy of the web).
E	(float): Young modulus.
t_p	(float): Panel zone thickness.
t_dp	(float, optional): Doubler plate thickness. Defaults to 0.0.
a_s	(float, optional): Strain hardening. Defaults to 0.03.
pinchx	(float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25.
pinchy	(float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75.
dmg1	(float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
dmg2	(float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
beta	(float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
safety_factor	(bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	d_c needs to be positive.
NegativeValue	bf_c needs to be positive.
NegativeValue	tf_c needs to be positive.
NegativeValue	d_b needs to be positive.
NegativeValue	tf_b needs to be positive.
NegativeValue	Fy needs to be positive.
NegativeValue	E needs to be positive.
NegativeValue	t_p needs to be positive.
NegativeValue	a_s needs to be positive.

Reimplemented in Gupta1999SteelIShape.

Definition at line 531 of file MaterialModels.py.

```
        t_dp = 0.0, a_s = 0.03, pinchx = 0.25, pinchy = 0.75, dmg1 = 0.0, dmg2 = 0.0, beta = 0.0, safety_factor = False):
        """
        Constructor of the class.

        @param ID (int): Unique material model ID.
        @param d_c (float): Column depth.
        @param bf_c (float): Column flange width.
        @param tf_c (float): Column flange thickness.
        @param I_c (float): Column moment of inertia (strong axis).
        @param d_b (float): Beam depth.
        @param tf_b (float): Beam flange thickness.
        @param Fy (float): Yield strength (if assume continous column, Fy of the web).
        @param E (float): Young modulus.
        @param t_p (float): Panel zone thickness.
        @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
        @param a_s (float, optional): Strain hardening. Defaults to 0.03.
        @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25.
        @param pinchy (float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75.
        @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
        @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
        @param beta (float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
        @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: d_c needs to be positive.
        @exception NegativeValue: bf_c needs to be positive.
        @exception NegativeValue: tf_c needs to be positive.
        @exception NegativeValue: d_b needs to be positive.
        @exception NegativeValue: tf_b needs to be positive.
        @exception NegativeValue: Fy needs to be positive.
        @exception NegativeValue: E needs to be positive.
        @exception NegativeValue: t_p needs to be positive.
        @exception NegativeValue: a_s needs to be positive.
        """
        # Check
        if ID < 1: raise NegativeValue()
        if d_c < 0: raise NegativeValue()
        if bf_c < 0: raise NegativeValue()
        if tf_c < 0: raise NegativeValue()
        if d_b < 0: raise NegativeValue()
        if tf_b < 0: raise NegativeValue()
        if Fy < 0: raise NegativeValue()
        if E < 0: raise NegativeValue()
        if t_p < 0: raise NegativeValue()
        if a_s < 0: raise NegativeValue()

        # Arguments
        self.ID = ID
        self.d_c = d_c
        self.bf_c = bf_c
        self.tf_c = tf_c
        self.I_c = I_c
        self.d_b = d_b
        self.tf_b = tf_b
```

```
        self.Fy = Fy
        self.E = E
        self.t_p = t_p
        self.t_dp = t_dp
        self.a_s = a_s
        self.pinchx = pinchx
        self.pinchy = pinchy
        self.dmg1 = dmg1
        self.dmg2 = dmg2
        self.beta = beta
        if safety_factor:
            self.Ry = 1.2
        else:
            self.Ry = 1.0

        # Initialized the parameters that are dependent from others
        self.beam_section_name_tag = "None"
        self.col_section_name_tag = "None"
        self.Initialized = False
        self.ReInit()


```

7.27.3 Member Function Documentation

7.27.3.1 CheckApplicability()

```
def CheckApplicability (
                self )
```

Implementation of the homonym abstract method.

See parent class MaterialModels for detailed information.

Reimplemented from MaterialModels.

Definition at line 722 of file MaterialModels.py.

```
00722     def CheckApplicability(self):
00723         """
00724         Implementation of the homonym abstract method.
00725         See parent class MaterialModels for detailed information.
00726         """
00727         Check = True
00728         # No checks
00729         if not Check:
00730             print("The validity of the equations is not fullfilled.")
00731             print("!!!!!!! WARNING !!!!!!! Check material model of Gupta 1999, ID=", self.ID)
00732             print("")
00733 
00734 
```

7.27.3.2 Hysteretic()

```
def Hysteretic (
                self )
```

Generate the material model Hysteretic (Gupta 1999) using the computed parameters.

See _Hysteretic function for more information.

Definition at line 735 of file MaterialModels.py.

```
    def Hysteretic(self):
        """
        Generate the material model Hysteretic (Gupta 1999) using the computed parameters.
        See _Hysteretic function for more information.
        """
        _Hysteretic(self.ID, self.M1y, self.gamma1_y, self.M2y, self.gamma2_y, self.M3y, self.gamma3_y,
            self.pinchx, self.pinchy, self.dmg1, self.dmg2, self.beta)
        self.Initialized = True
        self.UpdateStoredData()


```

7.27.3.3 ReInit()

```
def ReInit (
              self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 609 of file MaterialModels.py.

```
00609     def ReInit(self):
00610         """
00611         Implementation of the homonym abstract method.
00612         See parent class DataManagement for detailed information.
00613         """
00614         # Check applicability
00615         self.CheckApplicability()
00616 
00617         # Members
00618         if self.beam_section_name_tag != "None": self.beam_section_name_tag = self.beam_section_name_tag + " (modified)"
00619         if self.col_section_name_tag != "None": self.col_section_name_tag = self.col_section_name_tag + " (modified)"
00620 
00621         # Trilinear Parameters
00622         self.t_pz = self.t_p + self.t_dp
00623         self.Vy = 0.55 * self.Fy * self.Ry * self.d_c * self.t_pz # Yield Shear
00624         self.G = self.E/(2.0 * (1.0 + 0.30)) # Shear Modulus
00625         self.Ke = 0.95 * self.G * self.t_pz * self.d_c # Elastic Stiffness
00626         self.Kp = 0.95 * self.G * self.bf_c * (self.tf_c * self.tf_c) / self.d_b # Plastic Stiffness
00627 
00628         # Define Trilinear Equivalent Rotational Spring
00629         # Yield point for Trilinear Spring at gamma1_y
00630         self.gamma1_y = self.Vy / self.Ke
00631         self.M1y = self.gamma1_y * (self.Ke * self.d_b)
00632         # Second Point for Trilinear Spring at 4 * gamma1_y
00633         self.gamma2_y = 4.0 * self.gamma1_y
00634         self.M2y = self.M1y + (self.Kp * self.d_b) * (self.gamma2_y - self.gamma1_y)
00635         # Third Point for Trilinear Spring at 100 * gamma1_y
00636         self.gamma3_y = 100.0 * self.gamma1_y
00637         self.M3y = self.M2y + (self.a_s * self.Ke * self.d_b) * (self.gamma3_y - self.gamma2_y)
00638 
00639         # Data storage for loading/saving
00640         self.UpdateStoredData()
00641 
00642 
```

7.27.3.4 ShowInfo()

```
def ShowInfo (
              self,
              plot = False,
              block = False )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the material model. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 682 of file MaterialModels.py.

```
00682     def ShowInfo(self, plot = False, block = False):
00683         """
00684         Implementation of the homonym abstract method.
00685         See parent class DataManagement for detailed information.
00686 
00687         @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
00688         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
00689         """
00690         print("")
00691         print("Requested info for Gupta 1999 material model Parameters, ID = {}".format(self.ID))
00692         print("Sections associated, column: {} ".format(self.col_section_name_tag))
00693         print("Sections associated, beam: {} ".format(self.beam_section_name_tag))
00694         print("gamma1_y = {} rad".format(self.gamma1_y))
00695         print("gamma2_y = {} rad".format(self.gamma2_y))
00696         print("gamma3_y = {} rad".format(self.gamma3_y))
00697         print("M1y = {} kNm".format(self.M1y/kNm_unit))
00698         print("M2y = {} kNm".format(self.M2y/kNm_unit))
00699         print("M3y = {} kNm".format(self.M3y/kNm_unit))
00700         print("")
00701 
00702         if plot:
00703             # Data for plotting
00704             # Last point for plot
00705             gamma3_y_plot = 10.0 * self.gamma1_y
00706             M3y_plot = self.M2y + (self.a_s * self.Ke * self.d_b) * (gamma3_y_plot - self.gamma2_y)
00707 
00708             x_axis = np.array([0.0, self.gamma1_y, self.gamma2_y, gamma3_y_plot])
00709             y_axis = np.array([0.0, self.M1y, self.M2y, M3y_plot])/kNm_unit
00710 
00711             fig, ax = plt.subplots()
00712             ax.plot(x_axis, y_axis, 'k-')
00713 
00714             ax.set(xlabel='Rotation [rad]', ylabel='Moment [kNm]',
00715                 title='Gupta 1999 material model (ID={})'.format(self.ID))
00716             ax.grid()
00717 
00718             if block:
00719                 plt.show()
00720 
00721 
```

7.27.3.5 UpdateStoredData()

```
def UpdateStoredData (
              self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 643 of file MaterialModels.py.

```
00643     def UpdateStoredData(self):
00644         """
00645         Implementation of the homonym abstract method.
00646         See parent class DataManagement for detailed information.
00647         """
00648         self.data = [["INFO_TYPE", "Gupta1999"], # Tag for differentiating different data
00649             ["ID", self.ID],
00650             ["beam_section_name_tag", self.beam_section_name_tag],
00651             ["col_section_name_tag", self.col_section_name_tag],
00652             ["d_c", self.d_c],
00653             ["bf_c", self.bf_c],
```

```
            ["tf_c", self.tf_c],
            ["I_c", self.I_c],
            ["d_b", self.d_b],
            ["tf_b", self.tf_b],
            ["Fy", self.Fy],
            ["E", self.E],
            ["G", self.G],
            ["t_p", self.t_p],
            ["t_dp", self.t_dp],
            ["t_pz", self.t_pz],
            ["a_s", self.a_s],
            ["pinchx", self.pinchx],
            ["pinchy", self.pinchy],
            ["dmg1", self.dmg1],
            ["dmg2", self.dmg2],
            ["beta", self.beta],
            ["Ry", self.Ry],
            ["Vy", self.Vy],
            ["Ke", self.Ke],
            ["Kp", self.Kp],
            ["gamma1_y", self.gamma1_y],
            ["M1y", self.M1y],
            ["gamma2_y", self.gamma2_y],
            ["M2y", self.M2y],
            ["gamma3_y", self.gamma3_y],
            ["M3y", self.M3y],
            ["Initialized", self.Initialized]]

```

7.27.4 Member Data Documentation

7.27.4.1 a_s

`a_s`

Definition at line 590 of file MaterialModels.py.

7.27.4.2 beam_section_name_tag

`beam_section_name_tag`

Definition at line 602 of file MaterialModels.py.

7.27.4.3 beta

`beta`

Definition at line 595 of file MaterialModels.py.

7.27.4.4 bf_c

```
bf_c
```

Definition at line 581 of file MaterialModels.py.

7.27.4.5 col_section_name_tag

```
col_section_name_tag
```

Definition at line 603 of file MaterialModels.py.

7.27.4.6 d_b

```
d_b
```

Definition at line 584 of file MaterialModels.py.

7.27.4.7 d_c

```
d_c
```

Definition at line 580 of file MaterialModels.py.

7.27.4.8 data

```
data
```

Definition at line 648 of file MaterialModels.py.

7.27.4.9 dmg1

```
dmg1
```

Definition at line 593 of file MaterialModels.py.

7.27.4.10 dmg2

```
dmg2
```

Definition at line 594 of file MaterialModels.py.

7.27.4.11 E

```
E
```

Definition at line 587 of file MaterialModels.py.

7.27.4.12 Fy

```
Fy
```

Definition at line 586 of file MaterialModels.py.

7.27.4.13 G

```
G
```

Definition at line 624 of file MaterialModels.py.

7.27.4.14 gamma1_y

```
gamma1_y
```

Definition at line 630 of file MaterialModels.py.

7.27.4.15 gamma2_y

```
gamma2_y
```

Definition at line 633 of file MaterialModels.py.

7.27.4.16 gamma3_y

```
gamma3_y
```

Definition at line 636 of file MaterialModels.py.

7.27.4.17 I_c

```
I_c
```

Definition at line 583 of file MaterialModels.py.

7.27.4.18 ID

```
ID
```

Definition at line 579 of file MaterialModels.py.

7.27.4.19 Initialized

```
Initialized
```

Definition at line 604 of file MaterialModels.py.

7.27.4.20 Ke

```
Ke
```

Definition at line 625 of file MaterialModels.py.

7.27.4.21 Kp

```
Kp
```

Definition at line 626 of file MaterialModels.py.

7.27.4.22 M1y

```
M1y
```

Definition at line 631 of file MaterialModels.py.

7.27.4.23 M2y

```
M2y
```

Definition at line 634 of file MaterialModels.py.

7.27.4.24 M3y

```
M3y
```

Definition at line 637 of file MaterialModels.py.

7.27.4.25 pinchx

```
pinchx
```

Definition at line 591 of file MaterialModels.py.

7.27.4.26 pinchy

```
pinchy
```

Definition at line 592 of file MaterialModels.py.

7.27.4.27 Ry

```
Ry
```

Definition at line 597 of file MaterialModels.py.

7.27.4.28 t_dp

`t_dp`

Definition at line 589 of file MaterialModels.py.

7.27.4.29 t_p

`t_p`

Definition at line 588 of file MaterialModels.py.

7.27.4.30 t_pz

`t_pz`

Definition at line 622 of file MaterialModels.py.

7.27.4.31 tf_b

`tf_b`

Definition at line 585 of file MaterialModels.py.

7.27.4.32 tf_c

`tf_c`

Definition at line 582 of file MaterialModels.py.

7.27.4.33 Vy

`Vy`

Definition at line 623 of file MaterialModels.py.

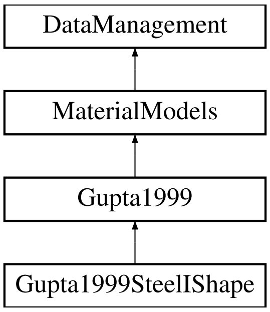
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.28 Gupta1999SteelIShape Class Reference

Class that is the children of Gupta1999 and combine the class SteelIShape (section) to retrieve the information needed.

Inheritance diagram for Gupta1999SteelIShape:



Public Member Functions

- def __init__ (self, int ID, SteelIShape col, SteelIShape beam, t_dp=0.0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0.0, dmg2=0.0, beta=0.0, safety_factor=False)

    Constructor of the class.

Public Attributes

- beam
- beam_section_name_tag
- col
- col_section_name_tag

7.28.1 Detailed Description

Class that is the children of Gupta1999 and combine the class SteelIShape (section) to retrieve the information needed.

Parameters

Gupta1999	Parent class.

Definition at line 746 of file MaterialModels.py.

7.28.2 Constructor & Destructor Documentation

7.28.2.1 __init__()

```
def __init__ (
                self,
                int ID,
                SteelIShape col,
                SteelIShape beam,
                t_dp = 0.0,
                a_s = 0.03,
                pinchx = 0.25,
                pinchy = 0.75,
                dmg1 = 0.0,
                dmg2 = 0.0,
                beta = 0.0,
                safety_factor = False )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class SteelIShape. The copy of the sections (col and beam) passed is stored in the member variable self.section.

Parameters

ID	(int): Unique material model ID.
col	(SteelIShape): SteelIShape column section object.
beam	(SteelIShape): SteelIShape beam section object.
t_dp	(float, optional): Doubler plate thickness. Defaults to 0.0.
a_s	(float, optional): Strain hardening. Defaults to 0.03.
pinchx	(float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25.
pinchy	(float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75
dmg1	(float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
dmg2	(float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
beta	(float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
safety_factor	(bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.

Reimplemented from Gupta1999.

Definition at line 752 of file MaterialModels.py.

```
00753         t_dp = 0.0, a_s = 0.03, pinchx = 0.25, pinchy = 0.75, dmg1 = 0.0, dmg2 = 0.0, beta = 0.0,
       safety_factor = False):
00754         """
00755         Constructor of the class. It passes the arguments into the parent class to generate the
       combination of the parent class
00756             and the section class SteelIShape.
00757         The copy of the sections (col and beam) passed is stored in the member variable self.section.
00758 
00759         @param ID (int): Unique material model ID.
00760         @param col (SteelIShape): SteelIShape column section object.
00761         @param beam (SteelIShape): SteelIShape beam section object.
00762         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
00763         @param a_s (float, optional): Strain hardening. Defaults to 0.03.
00764         @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading.
       Defaults to 0.25.
00765         @param pinchy (float, optional): Pinching factor for stress (or force) during reloading.
       Defaults to 0.75
00766         @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
00767         @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
00768         @param beta (float, optional): Power used to determine the degraded unloading stiffness based
       on ductility, mu-beta. Defaults to 0.0.
```

```
00769         @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters
      are used (not test results). Defaults to False.
00770         """
00771         self.col = deepcopy(col)
00772         self.beam = deepcopy(beam)
00773         super().__init__(ID, col.d, col.bf, col.tf, col.Iy, beam.d, beam.tf, col.Fy_web, col.E,
      col.tw,
00774             t_dp, a_s, pinchx, pinchy, dmg1, dmg2, beta, safety_factor)
00775         self.beam_section_name_tag = beam.name_tag
00776         self.col_section_name_tag = col.name_tag
00777         self.UpdateStoredData()
00778
00779
```

7.28.3 Member Data Documentation

7.28.3.1 beam

`beam`

Definition at line 772 of file MaterialModels.py.

7.28.3.2 beam_section_name_tag

`beam_section_name_tag`

Definition at line 775 of file MaterialModels.py.

7.28.3.3 col

`col`

Definition at line 771 of file MaterialModels.py.

7.28.3.4 col_section_name_tag

`col_section_name_tag`

Definition at line 776 of file MaterialModels.py.

The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.29 IDGenerator Class Reference

Class that manage the ID generation.

Public Member Functions

- def __init__ (self)
  The class constructor.
- def GenerateIDElement (self)
  Method that generate a unique element ID.
- def GenerateIDFiber (self)
  Method that generate a unique fiber ID.
- def GenerateIDMat (self)
  Method that generate a unique material ID.
- def GenerateIDNode (self)
  Method that generate a unique node ID.

Public Attributes

- current_element_ID
- current_fiber_ID
- current_mat_ID
- current_node_ID

7.29.1 Detailed Description

Class that manage the ID generation.

USE ONLY IF EVERY NODE IS DEFINED BY THE USER (because the OpenSeesPyAssistant modules use the convention defined in the functions above).

Definition at line 412 of file FunctionalFeatures.py.

7.29.2 Constructor & Destructor Documentation

7.29.2.1 __init__()

```
def __init__ (
              self )
```

The class constructor.

Definition at line 416 of file FunctionalFeatures.py.

```
00416     def __init__(self):
00417         """The class constructor.
00418         """
00419         self.current_node_ID = 0
00420         self.current_element_ID = 0
00421         self.current_mat_ID = 0
00422         self.current_fiber_ID = 0
00423
```

7.29.3 Member Function Documentation

7.29.3.1 GenerateIDElement()

```
def GenerateIDElement (
              self )
```

Method that generate a unique element ID.

Returns

int: The element ID.

Definition at line 433 of file FunctionalFeatures.py.

```
00433     def GenerateIDElement(self):
00434         """
00435         Method that generate a unique element ID.
00436 
00437         @returns int: The element ID.
00438         """
00439         self.current_element_ID = self.current_element_ID + 1
00440         return self.current_element_ID
00441 
```

7.29.3.2 GenerateIDFiber()

```
def GenerateIDFiber (
              self )
```

Method that generate a unique fiber ID.

Returns

int: The fiber ID.

Definition at line 451 of file FunctionalFeatures.py.

```
00451     def GenerateIDFiber(self):
00452         """
00453         Method that generate a unique fiber ID.
00454 
00455         @returns int: The fiber ID.
00456         """
00457         self.current_fiber_ID = self.current_fiber_ID + 1
00458         return self.current_fiber_ID
00459 
00460 
```

7.29.3.3 GenerateIDMat()

```
def GenerateIDMat (
              self )
```

Method that generate a unique material ID.

Returns

int: The material ID.

Definition at line 442 of file FunctionalFeatures.py.

```
00442     def GenerateIDMat(self):
00443         """
00444         Method that generate a unique material ID.
00445 
00446         @returns int: The material ID.
00447         """
00448         self.current_mat_ID = self.current_mat_ID + 1
00449         return self.current_mat_ID
00450 
```

7.29.3.4 GenerateIDNode()

```
def GenerateIDNode (
              self )
```

Method that generate a unique node ID.

Returns

int: The node ID.

Definition at line 424 of file FunctionalFeatures.py.

```
00424     def GenerateIDNode(self):
00425         """
00426         Method that generate a unique node ID.
00427 
00428         @returns int: The node ID.
00429         """
00430         self.current_node_ID = self.current_node_ID + 1
00431         return self.current_node_ID
00432 
```

7.29.4 Member Data Documentation

7.29.4.1 current_element_ID

```
current_element_ID
```

Definition at line 420 of file FunctionalFeatures.py.

7.29.4.2 current_fiber_ID

`current_fiber_ID`

Definition at line 422 of file FunctionalFeatures.py.

7.29.4.3 current_mat_ID

`current_mat_ID`

Definition at line 421 of file FunctionalFeatures.py.

7.29.4.4 current_node_ID

`current_node_ID`

Definition at line 419 of file FunctionalFeatures.py.

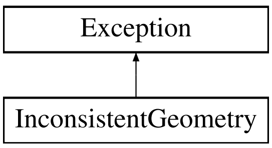
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/FunctionalFeatures.py

7.30 InconsistentGeometry Class Reference

Exception class for the "inconsistent geometry" error.

Inheritance diagram for InconsistentGeometry:



7.30.1 Detailed Description

Exception class for the "inconsistent geometry" error.

Definition at line 31 of file ErrorHandling.py.

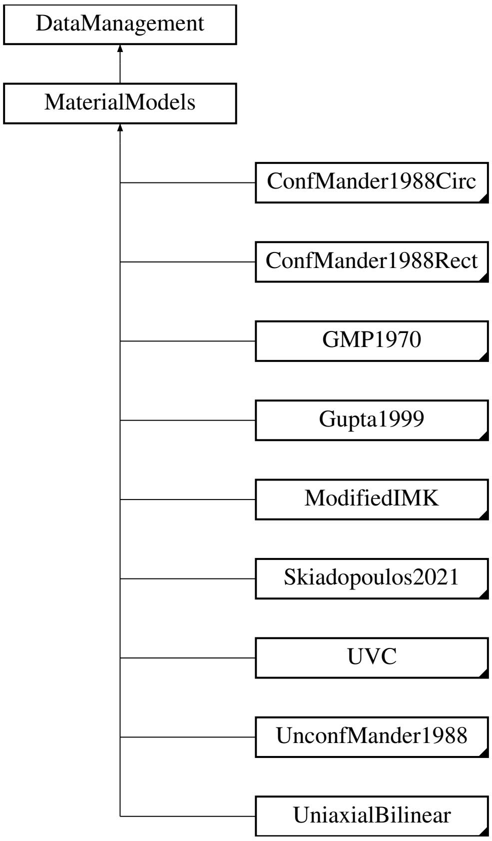
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py

7.31 MaterialModels Class Reference

Parent abstract class for the storage and manipulation of a material model's information (mechanical and geometrical parameters, etc) and initialisation in the model.

Inheritance diagram for MaterialModels:



Public Member Functions

- def CheckApplicability (self)

  Abstract function used to check the applicability of the material model.

7.31.1 Detailed Description

Parent abstract class for the storage and manipulation of a material model's information (mechanical and geometrical parameters, etc) and initialisation in the model.

Parameters

DataManagement	Parent abstract class.

Definition at line 19 of file MaterialModels.py.

7.31.2 Member Function Documentation

7.31.2.1 CheckApplicability()

```
def CheckApplicability (
              self )
```

Abstract function used to check the applicability of the material model.

Reimplemented in ModifiedIMK, Gupta1999, Skiadopoulos2021, UnconfMander1988, ConfMander1988Rect, ConfMander1988Circ, UniaxialBilinear, GMP1970, and UVC.

Definition at line 27 of file MaterialModels.py.

```
    def CheckApplicability(self):
        """
        Abstract function used to check the applicability of the material model.
        """
        pass


```

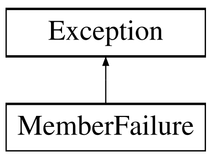
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.32 MemberFailure Class Reference

Exception class for the "member failure" error.

Inheritance diagram for MemberFailure:



7.32.1 Detailed Description

Exception class for the "member failure" error.

Definition at line 36 of file ErrorHandling.py.

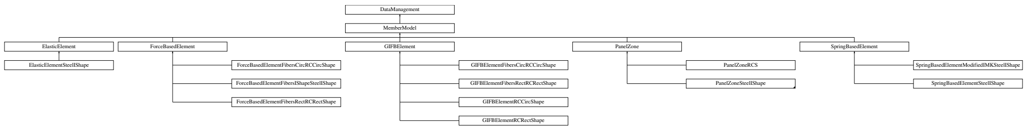
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py

7.33 MemberModel Class Reference

Parent abstract class for the storage and manipulation of a member's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Inheritance diagram for MemberModel:



Public Member Functions

- def Record (self, ele_ID, str name_txt, str data_dir, force_rec=True, def_rec=True, time_rec=True)

    Abstract method that records the forces, deformation and time of the member associated with the class.
- def RecordNodeDef (self, int iNode_ID, int jNode_ID, str name_txt, str data_dir, time_rec=True)

    Abstract method that records the deformation and time of the member's nodes associated with the class.

7.33.1 Detailed Description

Parent abstract class for the storage and manipulation of a member's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Parameters

DataManagement	Parent abstract class.

Definition at line 22 of file MemberModel.py.

7.33.2 Member Function Documentation

7.33.2.1 Record()

```
def Record (
             self,
             ele_ID,
            str name_txt,
            str data_dir,
             force_rec = True,
             def_rec = True,
             time_rec = True )
```

Abstract method that records the forces, deformation and time of the member associated with the class.

Parameters

ele_ID	(int): The ID of the element that will be recorded.
name_txt	(str): Name of the recorded data (no .txt).
data_dir	(str): Directory for the storage of data.
force_rec	(bool, optional): Option to record the forces (Fx, Fy, Mz). Defaults to True.
def_rec	(bool, optional): Option to record the deformation (theta) for ZeroLength element. Defaults to True.
time_rec	(bool, optional): Option to record time. Defaults to True.

Reimplemented in PanelZone, ElasticElement, ForceBasedElement, GIFBElement, and SpringBasedElement.

Definition at line 29 of file MemberModel.py.

```
00029     def Record(self, ele_ID, name_txt: str, data_dir: str, force_rec = True, def_rec = True, time_rec
       = True):
00030         """
00031         Abstract method that records the forces, deformation and time of the member associated with
       the class.
00032 
00033         @param ele_ID (int): The ID of the element that will be recorded.
00034         @param name_txt (str): Name of the recorded data (no .txt).
00035         @param data_dir (str): Directory for the storage of data.
00036         @param force_rec (bool, optional): Option to record the forces (Fx, Fy, Mz). Defaults to True.
00037         @param def_rec (bool, optional): Option to record the deformation (theta) for ZeroLength
       element. Defaults to True.
00038         @param time_rec (bool, optional): Option to record time. Defaults to True.
00039         """
00040         if self.Initialized:
00041             if not os.path.exists(data_dir):
00042                 print("Folder {} not found in this directory; creating one".format(data_dir))
00043                 os.makedirs(data_dir)
00044 
00045             if time_rec:
00046                 if force_rec:
00047                     recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-time",
       "-ele", ele_ID, "force")
00048                 if def_rec:
00049                     recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-time",
       "-ele", ele_ID, "deformation")
00050             else:
00051                 if force_rec:
00052                     recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-ele",
       ele_ID, "force")
00053                 if def_rec:
00054                     recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-ele",
       ele_ID, "deformation")
00055         else:
00056                 print("The element is not initialized (node and/or elements not created), ID =
       {}".format(ele_ID))
00057 
```

7.33.2.2 RecordNodeDef()

```
def RecordNodeDef (
                self,
                int iNode_ID,
                int jNode_ID,
                str name_txt,
                str data_dir,
                time_rec = True )
```

Abstract method that records the deformation and time of the member's nodes associated with the class.

Parameters

iNode_ID	(int): ID of the node i.

Parameters

jNode_ID	(int): ID of the node j.
name_txt	(str): Name of the recorded data (no .txt).
data_dir	(str): Directory for the storage of data.
time_rec	(bool, optional): Option to record time. Defaults to True.

Reimplemented in PanelZone, ElasticElement, SpringBasedElement, ForceBasedElement, and GIFBElement.

Definition at line 59 of file MemberModel.py.

```
00059     def RecordNodeDef(self, iNode_ID: int, jNode_ID: int, name_txt: str, data_dir: str, time_rec =
      True):
00060         """
00061         Abstract method that records the deformation and time of the member’s nodes associated with
      the class.
00062
00063         @param iNode_ID (int): ID of the node i.
00064         @param jNode_ID (int): ID of the node j.
00065         @param name_txt (str): Name of the recorded data (no .txt).
00066         @param data_dir (str): Directory for the storage of data.
00067         @param time_rec (bool, optional): Option to record time. Defaults to True.
00068         """
00069         if self.Initialized:
00070             if not os.path.exists(data_dir):
00071                 print("Folder {} not found in this directory; creating one".format(data_dir))
00072                 os.makedirs(data_dir)
00073
00074             if time_rec:
00075                 recorder("Node", "-file", ’{}/{}.txt’.format(data_dir, name_txt), "-time", "-node",
      iNode_ID, jNode_ID, "-dof", 1, 2, 3, "disp")
00076             else:
00077                 recorder("Node", "-file", ’{}/{}.txt’.format(data_dir, name_txt), "-node", iNode_ID,
      jNode_ID, "-dof", 1, 2, 3, "disp")
00078         else:
00079                 print("The element is not initialized (node and/or elements not created), iNode ID =
      {}, jNode ID = {}".format(iNode_ID, jNode_ID))
00080
00081
```

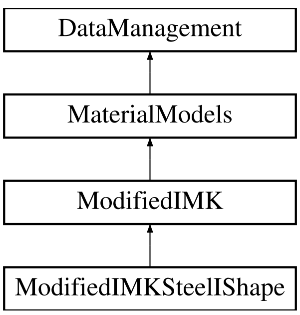
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.34 ModifiedIMK Class Reference

Class that stores funcions and material properties of a steel double symmetric I-shape profile with modified Ibarra-Medina-Krawinkler as the material model for the nonlinear springs and the OpenSeesPy command type used to model it is Bilin.

Inheritance diagram for ModifiedIMK:



Public Member Functions

- def __init__ (self, int ID, str Type, d, bf, tf, tw, h_1, Iy_mod, iz, E, Fy, Npl, My, L, N_G=0, K_factor=3, L_0=-1, L_b=-1, Mc=-1, K=-1, theta_u=-1, safety_factors=False)

  Constructor of the class.
- def Bilin (self)

  Generate the material model Bilin (Modified IMK) using the computed parameters.
- def CheckApplicability (self)

  Implementation of the homonym abstract method.
- def Computea (self)

  Method that computes the strain hardening ratio with the n modification.
- def Computea_s (self)

  Method that computes the modified strain hardening ratio for the spring.
- def ComputeK (self)

  Method that computes the residual strength ratio.
- def ComputeKe (self)

  Method that computes the elastic stiffness.
- def ComputeMc (self)

  Method that computes the capping moment.
- def ComputeMyStar (self)

  Method that computes the effective yield moment.
- def ComputeRefEnergyDissipationCap (self)

  Method that computes the reference energy dissipation capacity.
- def ComputeTheta_p (self)

  Method that computes the plastic rotation.
- def ComputeTheta_pc (self)

  Method that computes the post capping rotation.
- def ComputeTheta_u (self)

  Method that computes the ultimate rotation.
- def ComputeTheta_y (self)

  Method that computes the yield rotation.
- def ReInit (self, Mc=-1, K=-1, theta_u=-1)

  Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False)

  Implementation of the homonym abstract method.
- def UpdateStoredData (self)

  Implementation of the homonym abstract method.

Public Attributes

- a
- a_s
- bf
- d
- data
- E
- Fy
- gamma_rm
- h_1
- ID
- Initialized

- Iy_mod
- iz
- K
- K_factor
- Ke
- L
- L_0
- L_b
- Mc
- McMy
- My
- My_star
- N_G
- Npl
- prob_factor
- rate_det
- section_name_tag
- tf
- theta_p
- theta_pc
- theta_u
- theta_y
- tw
- Type

Static Public Attributes

- float n = 10.0

7.34.1 Detailed Description

Class that stores funcions and material properties of a steel double symmetric I-shape profile with modified Ibarra-Medina-Krawinkler as the material model for the nonlinear springs and the OpenSeesPy command type used to model it is Bilin.

The default values are valid for a simple cantelever. For more information about the empirical model for the computation of the parameters, see Lignos Krawinkler 2011. The parameter 'n' is used as global throughout the SteelIShape sections to optimise the program (given the fact that is constant everytime).

Parameters

MaterialModels	Parent abstract class.

Definition at line 34 of file MaterialModels.py.

7.34.2 Constructor & Destructor Documentation

7.34.2.1 __init__()

```
def __init__ (
              self,
             int ID,
             str Type,
              d,
              bf,
              tf,
              tw,
              h_1,
              Iy_mod,
              iz,
              E,
              Fy,
              Npl,
              My,
              L,
              N_G = 0,
              K_factor = 3,
              L_0 = -1,
              L_b = -1,
              Mc = -1,
              K = -1,
              theta_u = -1,
              safety_factors = False )
```

Constructor of the class.

Every argument that is optional and is initialised as -1, will be computed in this class.

Parameters

ID	(int): ID of the material model.
Type	(str): Type of the section. It can be 'Col' for column or 'Beam' for beams.
d	(float): Depth of the section.
bf	(float): Flange's width of the section
tf	(float): Flange's thickness of the section
tw	(float): Web's thickness of the section
h_1	(float): Depth excluding the flange's thicknesses and the weld fillets.
Iy_mod	(float): n modified moment of inertia (strong axis)
iz	(float): Radius of gyration (weak axis).
E	(float): Young modulus.
Fy	(float): Yield strength.
Npl	(float): Maximal vertical axial load.
My	(float): Yielding moment.
L	(float): Effective length of the element associated with this section. If the panel zone is present, exclude its dimension.
N_G	(float, optional): Gravity axial load. Defaults to 0.
K_factor	(float, optional): Rigidity factor. Defaults to 3 (assuming cantilever).
L_0	(float, optional): Position of the inflection point. Defaults to -1, e.g. computed as the total length, assuming cantilever.
L_b	(float, optional): Maximal unbraced lateral torsional buckling length. Defaults to -1, e.g. computed as the total length, assuming cantilever with no bracing support.

Parameters

Mc	(float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
K	(float, optional): Residual strength ratio. Defaults to -1, e.g. computed in ComputeK.
theta_u	(float, optional): Ultimate rotation. Defaults to -1, e.g. computed in ComputeTheta_u.
safety_factors	(bool, optional): Safety factors used if standard mechanical parameters are used (not test results). Defaults to False.

Exceptions

NegativeValue	ID needs to be a positive integer.
WrongArgument	Type needs to be 'Col' or 'Beam'.
NegativeValue	d needs to be positive.
NegativeValue	bf needs to be positive.
NegativeValue	tf needs to be positive.
NegativeValue	tw needs to be positive.
NegativeValue	h_1 needs to be positive.
NegativeValue	Iy_mod needs to be positive.
NegativeValue	iz needs to be positive.
NegativeValue	E needs to be positive.
NegativeValue	Fy needs to be positive.
NegativeValue	Npl needs to be positive.
NegativeValue	My needs to be positive.
NegativeValue	L needs to be positive.
NegativeValue	N_G needs to be positive.
NegativeValue	L_0 needs to be positive if different from -1.
NegativeValue	L_b needs to be positive if different from -1.
NegativeValue	Mc needs to be positive if different from -1.
NegativeValue	K needs to be positive if different from -1.
NegativeValue	theta_u needs to be positive if different from -1.
InconsistentGeometry	h_1 can't be bigger than d
MemberFailure	N_G can't be bigger than Npl (section failure).
InconsistentGeometry	L_0 can't be bigger than L

Reimplemented in ModifiedIMKSteelIShape.

Definition at line 47 of file MaterialModels.py.

```
00048         N_G = 0, K_factor = 3, L_0 = -1, L_b = -1, Mc = -1, K = -1, theta_u = -1, safety_factors =
       False):
00049         """
00050         Constructor of the class. Every argument that is optional and is initialised as -1, will be
       computed in this class.
00051
00052         @param ID (int): ID of the material model.
00053         @param Type (str): Type of the section. It can be 'Col' for column or 'Beam' for beams.
00054         @param d (float): Depth of the section.
00055         @param bf (float): Flange's width of the section
00056         @param tf (float): Flange's thickness of the section
00057         @param tw (float): Web's thickness of the section
00058         @param h_1 (float): Depth excluding the flange's thicknesses and the weld fillets.
00059         @param Iy_mod (float): n modified moment of inertia (strong axis)
00060         @param iz (float): Radius of gyration (weak axis).
00061         @param E (float): Young modulus.
00062         @param Fy (float): Yield strength.
00063         @param Npl (float): Maximal vertical axial load.
00064         @param My (float): Yielding moment.
00065         @param L (float): Effective length of the element associated with this section.
```

```
            If the panel zone is present, exclude its dimension.
        @param N_G (float, optional): Gravity axial load. Defaults to 0.
        @param K_factor (float, optional): Rigidity factor. Defaults to 3 (assuming cantilever).
        @param L_0 (float, optional): Position of the inflection point.
            Defaults to -1, e.g. computed as the total length, assuming cantilever.
        @param L_b (float, optional): Maximal unbraced lateral torsional buckling length.
            Defaults to -1, e.g. computed as the total length, assuming cantilever with no bracing support.
        @param Mc (float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
        @param K (float, optional): Residual strength ratio. Defaults to -1, e.g. computed in ComputeK.
        @param theta_u (float, optional): Ultimate rotation. Defaults to -1, e.g. computed in ComputeTheta_u.
        @param safety_factors (bool, optional): Safety factors used if standard mechanical parameters are used (not test results). Defaults to False.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception WrongArgument: Type needs to be 'Col' or 'Beam'.
        @exception NegativeValue: d needs to be positive.
        @exception NegativeValue: bf needs to be positive.
        @exception NegativeValue: tf needs to be positive.
        @exception NegativeValue: tw needs to be positive.
        @exception NegativeValue: h_1 needs to be positive.
        @exception NegativeValue: Iy_mod needs to be positive.
        @exception NegativeValue: iz needs to be positive.
        @exception NegativeValue: E needs to be positive.
        @exception NegativeValue: Fy needs to be positive.
        @exception NegativeValue: Npl needs to be positive.
        @exception NegativeValue: My needs to be positive.
        @exception NegativeValue: L needs to be positive.
        @exception NegativeValue: N_G needs to be positive.
        @exception NegativeValue: L_0 needs to be positive if different from -1.
        @exception NegativeValue: L_b needs to be positive if different from -1.
        @exception NegativeValue: Mc needs to be positive if different from -1.
        @exception NegativeValue: K needs to be positive if different from -1.
        @exception NegativeValue: theta_u needs to be positive if different from -1.
        @exception InconsistentGeometry: h_1 can't be bigger than d
        @exception MemberFailure: N_G can't be bigger than Npl (section failure).
        @exception InconsistentGeometry: L_0 can't be bigger than L
        """
        # Check
        if ID < 0: raise NegativeValue()
        if Type != "Beam" and Type != "Col": raise WrongArgument()
        if d < 0: raise NegativeValue()
        if bf < 0: raise NegativeValue()
        if tf < 0: raise NegativeValue()
        if tw < 0: raise NegativeValue()
        if h_1 < 0: raise NegativeValue()
        if Iy_mod < 0: raise NegativeValue()
        if iz < 0: raise NegativeValue()
        if E < 0: raise NegativeValue()
        if Fy < 0: raise NegativeValue()
        if Npl < 0: raise NegativeValue()
        if My < 0: raise NegativeValue()
        if L < 0: raise NegativeValue()
        if N_G < 0: raise NegativeValue()
        if L_0 != -1 and L_0 < 0: raise NegativeValue()
        if L_b != -1 and L_b < 0: raise NegativeValue()
        if Mc != -1 and Mc < 0: raise NegativeValue()
        if K != -1 and K < 0: raise NegativeValue()
        if theta_u != -1 and theta_u < 0: raise NegativeValue()
        if h_1 > d: raise InconsistentGeometry()
        if N_G > Npl: raise MemberFailure()
        if L_0 > L: raise InconsistentGeometry()

        # Arguments
        self.Type = Type
        self.ID = ID
        self.d = d
        self.bf = bf
        self.tf = tf
        self.tw = tw
        self.h_1 = h_1
        self.Iy_mod = Iy_mod
        self.iz = iz
        self.E = E
        self.Fy = Fy
        self.Npl = Npl
        self.My = My
        self.L = L
        self.N_G = N_G
        self.K_factor = K_factor
        self.L_0 = L if L_0 == -1 else L_0
        self.L_b = L if L_b == -1 else L_b

        # Initialized the parameters that are dependent from others
        self.section_name_tag = "None"
```

```
        self.Initialized = False
        if safety_factors:
            self.gamma_rm = 1.25
            self.prob_factor = 1.15
        else:
            self.gamma_rm = 1.0
            self.prob_factor = 1.0
        self.ReInit(Mc, K, theta_u)


```

7.34.3 Member Function Documentation

7.34.3.1 Bilin()

```
def Bilin (
              self )
```

Generate the material model Bilin (Modified IMK) using the computed parameters.

See _Bilin function for more information.

Definition at line 478 of file MaterialModels.py.

```
00478     def Bilin(self):
00479         """
00480         Generate the material model Bilin (Modified IMK) using the computed parameters.
00481         See _Bilin function for more information.
00482         """
00483         _Bilin(self.ID, self.Ke, self.a_s, self.My_star, self.theta_p, self.theta_pc, self.K,
       self.theta_u, self.rate_det)
00484         self.Initialized = True
00485         self.UpdateStoredData()
00486
00487
```

7.34.3.2 CheckApplicability()

```
def CheckApplicability (
              self )
```

Implementation of the homonym abstract method.

See parent class MaterialModels for detailed information.

Reimplemented from MaterialModels.

Definition at line 289 of file MaterialModels.py.

```
00289     def CheckApplicability(self):
00290         """
00291         Implementation of the homonym abstract method.
00292         See parent class MaterialModels for detailed information.
00293         """
00294         Check = True
00295         if self.Type == "Beam":
00296             if self.d/self.tw < 20 or self.d/self.tw > 55:
00297                 Check = False
00298                 print("The d/tw check was not fullfilled")
00299             if self.L_b/self.iz < 20 or self.L_b/self.iz > 80:
00300                 Check = False
00301                 print("The Lb/iz check was not fullfilled")
00302             if self.bf/2/self.tf < 4 or self.bf/2/self.tf  > 8:
```

```
                Check = False
                print("The bf/2/tf check was not fullfilled")
            if self.L/self.d < 2.5 or self.L/self.d > 7:
                Check = False
                print("The check L/d was not fullfilled")
            if self.d < 102*mm_unit or self.d > 914*mm_unit:
                Check = False
                print("The d check was not fullfilled")
            if self.Fy < 240*MPa_unit or self.Fy > 450*MPa_unit:
                Check = False
                print("The Fy check was not fullfilled")
        else:
            if self.h_1/self.tw < 3.71 or self.d/self.tw > 57.5:
                Check = False
                print("The h1/tw check was not fullfilled")
            if self.L_b/self.iz < 38.4 or self.L_b/self.iz > 120:
                Check = False
                print("The Lb/iz check was not fullfilled")
            if self.N_G/self.Npl < 0 or self.N_G/self.Npl > 0.75:
                Check = False
                print("The NG/Npl check was not fullfilled")
        if not Check:
            print("The validity of the equations is not fullfilled.")
            print("!!!!!!! WARNING !!!!!!! Check material model of Modified IMK, ID=", self.ID)
            print("")

```

7.34.3.3 Computea()

def Computea (
    self)

Method that computes the strain hardening ratio with the n modification.

Returns

float: Strain hardening ratio.

Definition at line 337 of file MaterialModels.py.

```
00337     def Computea(self):
00338         """
00339         Method that computes the strain hardening ratio with the n modification.
00340 
00341         @returns float: Strain hardening ratio.
00342         """
00343         # strain hardening ratio of spring
00344         return (n+1.0)*self.My_star*(self.McMy-1.0)/(self.Ke*self.theta_p)
00345 
```

7.34.3.4 Computea_s()

def Computea_s (
    self)

Method that computes the modified strain hardening ratio for the spring.

For more info see Ibarra & Krawinkler 2005.

Returns

float: Strain hardening ratio.

Definition at line 346 of file MaterialModels.py.

```
00346     def Computea_s(self):
00347         """
00348         Method that computes the modified strain hardening ratio for the spring.
00349         For more info see Ibarra & Krawinkler 2005.
00350 
00351         @returns float: Strain hardening ratio.
00352         """
00353         return self.a/(1.0+n*(1.0-self.a))
00354 
```

7.34.3.5 ComputeK()

```
def ComputeK (
                self )
```

Method that computes the residual strength ratio.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

Returns

float: Residual strength ratio.

Definition at line 384 of file MaterialModels.py.

```
00384     def ComputeK(self):
00385         """
00386         Method that computes the residual strength ratio.
00387         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00388 
00389         @returns float: Residual strength ratio.
00390         """
00391         if self.Type == "Beam":
00392             return 0.4
00393         else:
00394             tmp = 0.5-0.4*self.N_G/self.Npl
00395             return max(tmp, 0)
00396 
```

7.34.3.6 ComputeKe()

```
def ComputeKe (
                self )
```

Method that computes the elastic stiffness.

Returns

float: The stiffness

Definition at line 329 of file MaterialModels.py.

```
00329     def ComputeKe(self):
00330         """
00331         Method that computes the elastic stiffness.
00332 
00333         @returns float: The stiffness
00334         """
00335         return self.K_factor*n*self.E*self.Iy_mod/self.L
00336 
```

7.34.3.7 ComputeMc()

```
def ComputeMc (
              self )
```

Method that computes the capping moment.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

Returns

float: Capping moment.

Definition at line 370 of file MaterialModels.py.

```
    def ComputeMc(self):
        """
        Method that computes the capping moment.
        For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

        @returns float: Capping moment.
        """
        if self.Type == "Beam":
            return self.My_star*1.11
            # For RBS: My_star*1.09
        else:
            tmp =
      12.5*(self.h_1/self.tw)**(-0.2)*(self.L_b/self.iz)**(-0.4)*(1-self.N_G/self.Npl)**0.4
            return max(min(1.3, tmp), 1.0)*self.My_star

```

7.34.3.8 ComputeMyStar()

```
def ComputeMyStar (
              self )
```

Method that computes the effective yield moment.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

Returns

float: Effective yield moment.

Definition at line 355 of file MaterialModels.py.

```
    def ComputeMyStar(self):
        """
        Method that computes the effective yield moment.
        For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

        @returns float: Effective yield moment.
        """
        if self.Type == "Beam":
            return self.prob_factor*self.My*self.gamma_rm*1.1
        else:
            if self.N_G/self.Npl > 0.2:
                return 1.15*self.prob_factor*self.My*self.gamma_rm*(1-self.N_G/self.Npl)*9.0/8.0
            else:
                return 1.15*self.prob_factor*self.My*self.gamma_rm*(1-self.N_G/2.0/self.Npl)

```

7.34.3.9 ComputeRefEnergyDissipationCap()

```
def ComputeRefEnergyDissipationCap (
              self )
```

Method that computes the reference energy dissipation capacity.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

Returns

float: Reference energy dissipation capacity.

Definition at line 456 of file MaterialModels.py.

```
    def ComputeRefEnergyDissipationCap(self):
        """
        Method that computes the reference energy dissipation capacity.
        For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

        @returns float: Reference energy dissipation capacity.
        """
        if self.Type == "Beam":
            if self.d < 533.0*mm_unit:
                return
    495.0*(self.h_1/self.tw)**(-1.34)*(self.bf/2.0/self.tf)**(-0.595)*(self.Fy/(355.0*MPa_unit))**(-0.360)
            else:
                return
    536.0*(self.h_1/self.tw)**(-1.26)*(self.bf/2.0/self.tf)**(-0.525)*(self.L_b/self.iz)**(-0.130)*(self.Fy/(355.0*MPa_unit
                # With RBS: ...
        else:
            if self.N_G/self.Npl > 0.35:
                tmp =
    268000.0*(self.h_1/self.tw)**(-2.30)*(self.L_b/self.iz)**(-1.130)*(1.0-self.N_G/self.Npl)**(1.19)
                return min(tmp, 3.0)
            else:
                tmp =
    25000.0*(self.h_1/self.tw)**(-2.14)*(self.L_b/self.iz)**(-0.53)*(1.0-self.N_G/self.Npl)**(4.92)
                return min(tmp, 3.0)
```

7.34.3.10 ComputeTheta_p()

```
def ComputeTheta_p (
              self )
```

Method that computes the plastic rotation.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

Returns

float: Plastic rotation.

Definition at line 406 of file MaterialModels.py.

```
00406     def ComputeTheta_p(self):
00407         """
00408         Method that computes the plastic rotation.
00409         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00410 
00411         @returns float: Plastic rotation.
00412         """
00413         if self.Type == "Beam":
00414             if self.d < 533.0*mm_unit:
00415                 return
       0.0865*(self.h_1/self.tw)**(-0.365)*(self.bf/2.0/self.tf)**(-0.14)*(self.L_0/self.d)**(0.34)*(self.d/(533.0*mm_unit))*
00416             else:
00417                 return
       0.318*(self.h_1/self.tw)**(-0.550)*(self.bf/2.0/self.tf)**(-0.345)*(self.L_0/self.d)**(0.090)*(self.L_b/self.iz)**(-0.
00418                 # With RBS: ...
00419         else:
00420             tmp =
       294.0*(self.h_1/self.tw)**(-1.7)*(self.L_b/self.iz)**(-0.7)*(1.0-self.N_G/self.Npl)**(1.6) #
       *(self.E/self.Fy/gamma_rm)**(0.2) # EC8
00421             if tmp > 0.2:
00422                 tmp = 0.2
00423             # if tmp > self.theta_u-self.theta_y:
00424             #     tmp = (self.theta_u-self.theta_y)*0.799 # convergence issue
00425             return tmp
00426 
```

7.34.3.11 ComputeTheta_pc()

```
def ComputeTheta_pc (
                self )
```

Method that computes the post capping rotation.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

Returns

float: Post capping rotation.

Definition at line 427 of file MaterialModels.py.

```
00427     def ComputeTheta_pc(self):
00428         """
00429         Method that computes the post capping rotation.
00430         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00431 
00432         @returns float: Post capping rotation.
00433         """
00434         if self.Type == "Beam":
00435             if self.d < 533.0*mm_unit:
00436                 return
       5.63*(self.h_1/self.tw)**(-0.565)*(self.bf/2.0/self.tf)**(-0.800)*(self.d/(533.0*mm_unit))**(-0.280)*(self.Fy/(355.0*MP
00437             else:
00438                 return
       7.50*(self.h_1/self.tw)**(-0.610)*(self.bf/2.0/self.tf)**(-0.710)*(self.L_b/self.iz)**(-0.110)*(self.d/(533.0*mm_unit))
00439                 # With RBS: ...
00440         else:
00441             tmp =
       90.0*(self.h_1/self.tw)**(-0.8)*(self.L_b/self.iz)**(-0.8)*(1.0-self.N_G/self.Npl)**(2.5) #
       *(self.E/self.Fy/gamma_rm)**(0.07) # EC8
00442             return min(tmp, 0.3)
00443 
```

7.34.3.12 ComputeTheta_u()

```
def ComputeTheta_u (
                self )
```

Method that computes the ultimate rotation.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

Returns

float: Ultimate rotation.

Definition at line 444 of file MaterialModels.py.

```
00444     def ComputeTheta_u(self):
00445         """
00446         Method that computes the ultimate rotation.
00447         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00448 
00449         @returns float: Ultimate rotation.
00450         """
00451         if self.Type == "Beam":
00452             return 0.2
00453         else:
00454             return 0.15
00455 
```

7.34.3.13 ComputeTheta_y()

```
def ComputeTheta_y (
                self )
```

Method that computes the yield rotation.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

Returns

float: Yield rotation.

Definition at line 397 of file MaterialModels.py.

```
00397     def ComputeTheta_y(self):
00398         """
00399         Method that computes the yield rotation.
00400         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00401 
00402         @returns float: Yield rotation.
00403         """
00404         return self.My_star/self.Ke*(n+1)
00405 
```

7.34.3.14 ReInit()

```
def ReInit (
                self,
                Mc = -1,
                K = -1,
                theta_u = -1 )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

Mc	(float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
K	(float, optional): Residual strength ratio. Defaults to -1, e.g. computed in ComputeK.
theta←_u	(float, optional): Ultimate rotation. Defaults to -1, e.g. computed in ComputeTheta_u.

Definition at line 160 of file MaterialModels.py.

```
00160     def ReInit(self, Mc = -1, K = -1, theta_u = -1):
00161         """
00162         Implementation of the homonym abstract method.
00163         See parent class DataManagement for detailed information.
00164
00165         @param Mc (float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
00166         @param K (float, optional): Residual strength ratio. Defaults to -1, e.g. computed in
       ComputeK.
00167         @param theta_u (float, optional): Ultimate rotation. Defaults to -1, e.g. computed in
       ComputeTheta_u.
00168         """
00169         # Precompute some members
00170         self.My_star = self.ComputeMyStar()
00171
00172         # Arguments
00173         self.Mc = self.ComputeMc() if Mc == -1 else Mc
00174         self.K = self.ComputeK() if K == -1 else K
00175         self.theta_u = self.ComputeTheta_u() if theta_u == -1 else theta_u
00176
00177         # Check applicability
00178         self.CheckApplicability()
00179
00180         # Members
00181         self.Ke = self.ComputeKe()
00182         self.theta_y = self.ComputeTheta_y()
00183         self.theta_p = self.ComputeTheta_p()
00184         self.theta_pc = self.ComputeTheta_pc()
00185         self.McMy = self.Mc/self.My_star
00186         self.rate_det = self.ComputeRefEnergyDissipationCap()
00187         self.a = self.Computea()
00188         self.a_s = self.Computea_s()
00189         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
       (modified)"
00190
00191         # Data storage for loading/saving
00192         self.UpdateStoredData()
00193
00194
```

7.34.3.15 ShowInfo()

```
def ShowInfo (
        self,
        plot = False,
        block = False )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the material model. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 237 of file MaterialModels.py.

```
00237     def ShowInfo(self, plot = False, block = False):
00238         """
00239         Implementation of the homonym abstract method.
00240         See parent class DataManagement for detailed information.
00241 
00242         @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
00243         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
00244         """
00245         Mr = self.K*self.My_star
00246         theta_p_plot = self.theta_p
00247         if self.theta_p > self.theta_u-self.theta_y:
00248             theta_p_plot = self.theta_u-self.theta_y
00249         theta_r = self.theta_y + theta_p_plot + self.theta_pc*(1.0-Mr/self.Mc)
00250         if theta_r > self.theta_u:
00251             theta_r = self.theta_u
00252             Mr = self.Mc*(1.0-1.0/self.theta_pc*(self.theta_u-self.theta_y-theta_p_plot))
00253 
00254         print("")
00255         print("Requested info for Modified IMK (Ibarra-Medina-Krawinkler) material model Parameters, ID = {}".format(self.ID))
00256         print("Section associated: {}".format(self.section_name_tag))
00257         print('theta y = {}'.format(self.theta_y))
00258         print('theta p = {}'.format(self.theta_p))
00259         print('theta r = {}'.format(theta_r))
00260         print('theta pc = {}'.format(self.theta_pc))
00261         print('theta u = {}'.format(self.theta_u))
00262         print('My star = {} kNm'.format(self.My_star/kNm_unit))
00263         print('Mc = {} kNm'.format(self.Mc/kNm_unit))
00264         print('Mr = {} kNm'.format(Mr/kNm_unit))
00265         print('a = {} '.format(self.a))
00266         print('as = {} '.format(self.a_s))
00267         print('lambda (deterioration rate) = {} '.format(self.rate_det))
00268         print("")
00269 
00270         if plot:
00271             # Data for plotting
00272             x_axis = np.array([0.0, self.theta_y, self.theta_y + theta_p_plot, theta_r, self.theta_u, self.theta_u])
00273             x_axis2 = np.array([self.theta_y + theta_p_plot, self.theta_y + theta_p_plot + self.theta_pc])
00274             y_axis = np.array([0.0, self.My_star, self.Mc, Mr, Mr, 0.0])/kNm_unit
00275             y_axis2 = np.array([self.Mc, 0.0])/kNm_unit
00276 
00277             fig, ax = plt.subplots()
00278             ax.plot(x_axis, y_axis, 'k-')
00279             ax.plot(x_axis2, y_axis2, 'k--')
00280 
00281             ax.set(xlabel='Rotation [rad]', ylabel='Moment [kNm]',
00282                 title='Modified IMK deterioration model (ID={})'.format(self.ID))
00283             ax.grid()
00284 
00285             if block:
00286                 plt.show()
00287 
00288 
```

7.34.3.16 UpdateStoredData()

```
def UpdateStoredData (
            self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 195 of file MaterialModels.py.

```
00195     def UpdateStoredData(self):
00196         """
00197         Implementation of the homonym abstract method.
00198         See parent class DataManagement for detailed information.
00199         """
00200         self.data = [["INFO_TYPE", "ModifiedIMK"], # Tag for differentiating different data
00201             ["ID", self.ID],
```

```
                ["section_name_tag", self.section_name_tag],
                ["Type", self.Type],
                ["d", self.d],
                ["bf", self.bf],
                ["tf", self.tf],
                ["tw", self.tw],
                ["h_1", self.h_1],
                ["Iy_mod", self.Iy_mod],
                ["iz", self.iz],
                ["E", self.E],
                ["Fy", self.Fy],
                ["L", self.L],
                ["N_G", self.N_G],
                ["K_factor", self.K_factor],
                ["Ke", self.Ke],
                ["L_0", self.L_0],
                ["L_b", self.L_b],
                ["gamma_rm", self.gamma_rm],
                ["prob_factor", self.prob_factor],
                ["Npl", self.Npl],
                ["My", self.My],
                ["My_star", self.My_star],
                ["Mc", self.Mc],
                ["McMy", self.McMy],
                ["K", self.K],
                ["theta_y", self.theta_y],
                ["theta_p", self.theta_p],
                ["theta_pc", self.theta_pc],
                ["theta_u", self.theta_u],
                ["rate_det", self.rate_det],
                ["a", self.a],
                ["a_s", self.a_s],
                ["Initialized", self.Initialized]]


```

7.34.4 Member Data Documentation

7.34.4.1 a

`a`

Definition at line 187 of file MaterialModels.py.

7.34.4.2 a_s

`a_s`

Definition at line 188 of file MaterialModels.py.

7.34.4.3 bf

`bf`

Definition at line 131 of file MaterialModels.py.

7.34.4.4 d

```
d
```

Definition at line 130 of file MaterialModels.py.

7.34.4.5 data

```
data
```

Definition at line 200 of file MaterialModels.py.

7.34.4.6 E

```
E
```

Definition at line 137 of file MaterialModels.py.

7.34.4.7 Fy

```
Fy
```

Definition at line 138 of file MaterialModels.py.

7.34.4.8 gamma_rm

```
gamma_rm
```

Definition at line 151 of file MaterialModels.py.

7.34.4.9 h_1

```
h_1
```

Definition at line 134 of file MaterialModels.py.

7.34.4.10 ID

```
ID
```

Definition at line 129 of file MaterialModels.py.

7.34.4.11 Initialized

```
Initialized
```

Definition at line 149 of file MaterialModels.py.

7.34.4.12 Iy_mod

```
Iy_mod
```

Definition at line 135 of file MaterialModels.py.

7.34.4.13 iz

```
iz
```

Definition at line 136 of file MaterialModels.py.

7.34.4.14 K

```
K
```

Definition at line 174 of file MaterialModels.py.

7.34.4.15 K_factor

```
K_factor
```

Definition at line 143 of file MaterialModels.py.

7.34.4.16 Ke

`Ke`

Definition at line 181 of file MaterialModels.py.

7.34.4.17 L

`L`

Definition at line 141 of file MaterialModels.py.

7.34.4.18 L_0

`L_0`

Definition at line 144 of file MaterialModels.py.

7.34.4.19 L_b

`L_b`

Definition at line 145 of file MaterialModels.py.

7.34.4.20 Mc

`Mc`

Definition at line 173 of file MaterialModels.py.

7.34.4.21 McMy

`McMy`

Definition at line 185 of file MaterialModels.py.

7.34.4.22 My

```
My
```

Definition at line 140 of file MaterialModels.py.

7.34.4.23 My_star

```
My_star
```

Definition at line 170 of file MaterialModels.py.

7.34.4.24 n

```
float n = 10.0 [static]
```

Definition at line 45 of file MaterialModels.py.

7.34.4.25 N_G

```
N_G
```

Definition at line 142 of file MaterialModels.py.

7.34.4.26 Npl

```
Npl
```

Definition at line 139 of file MaterialModels.py.

7.34.4.27 prob_factor

```
prob_factor
```

Definition at line 152 of file MaterialModels.py.

7.34.4.28 rate_det

```
rate_det
```

Definition at line 186 of file MaterialModels.py.

7.34.4.29 section_name_tag

```
section_name_tag
```

Definition at line 148 of file MaterialModels.py.

7.34.4.30 tf

```
tf
```

Definition at line 132 of file MaterialModels.py.

7.34.4.31 theta_p

```
theta_p
```

Definition at line 183 of file MaterialModels.py.

7.34.4.32 theta_pc

```
theta_pc
```

Definition at line 184 of file MaterialModels.py.

7.34.4.33 theta_u

```
theta_u
```

Definition at line 175 of file MaterialModels.py.

7.34.4.34 theta_y

```
theta_y
```

Definition at line 182 of file MaterialModels.py.

7.34.4.35 tw

```
tw
```

Definition at line 133 of file MaterialModels.py.

7.34.4.36 Type

```
Type
```

Definition at line 128 of file MaterialModels.py.

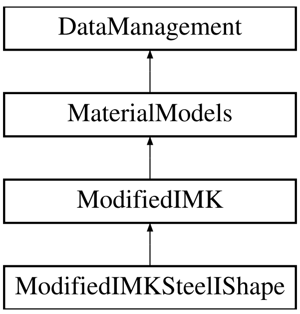
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.35 ModifiedIMKSteelIShape Class Reference

Class that is the children of ModifiedIMK and combine the class SteelIShape (section) to retrieve the information needed.

Inheritance diagram for ModifiedIMKSteelIShape:



Public Member Functions

- def __init__ (self, ID, SteelIShape section, N_G=0, K_factor=3, L_0=-1, L_b=-1, Mc=-1, K=-1, theta_u=-1, safety_factors=False)

  Constructor of the class.

Public Attributes

- section
- section_name_tag

Additional Inherited Members

7.35.1 Detailed Description

Class that is the children of ModifiedIMK and combine the class SteelIShape (section) to retrieve the information needed.

Parameters

ModifiedIMK	Parent class.

Definition at line 488 of file MaterialModels.py.

7.35.2 Constructor & Destructor Documentation

7.35.2.1 __init__()

```
def __init__ (
              self,
              ID,
             SteelIShape section,
              N_G = 0,
              K_factor = 3,
              L_0 = -1,
              L_b = -1,
              Mc = -1,
              K = -1,
              theta_u = -1,
              safety_factors = False )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class SteelIShape. Every argument that is optional and is initialised as -1, will be computed in this class. The copy of the section passed is stored in the member variable self.section.

Parameters

ID	(int): ID of the material model.
section	(SteelIShape): Object that store informations for a steel I shpae section.

Parameters

N_G	(float, optional): Gravity axial load. Defaults to 0.
K_factor	(float, optional): Rigidity factor. Defaults to 3 (assuming cantilever).
L_0	(float, optional): Position of the inflection point. Defaults to -1, e.g. computed as the total length, assuming cantilever.
L_b	(float, optional):Maximal unbraced lateral torsional buckling length. Defaults to -1, e.g computed as the total length, assuming cantilever with no bracing support.
Mc	(float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
K	(float, optional): Residual strength ratio. Defaults to -1, e.g. computed in ComputeK.
theta_u	(float, optional): Ultimate rotation. Defaults to -1, e.g. computed in ComputeTheta_u.
safety_factors	(bool, optional): Safety factors used if standard mechanical parameters are used (not test results). Defaults to False.

Reimplemented from ModifiedIMK.

Definition at line 494 of file MaterialModels.py.

```
00494     def __init__(self, ID, section: SteelIShape, N_G = 0, K_factor = 3, L_0 = -1, L_b = -1, Mc = -1, K = -1, theta_u = -1, safety_factors = False):
00495         """
00496         Constructor of the class. It passes the arguments into the parent class to generate the combination of the parent class
00497             and the section class SteelIShape.
00498         Every argument that is optional and is initialised as -1, will be computed in this class.
00499         The copy of the section passed is stored in the member variable self.section.
00500 
00501         @param ID (int): ID of the material model.
00502         @param section (SteelIShape): Object that store informations for a steel I shpae section.
00503         @param N_G (float, optional): Gravity axial load. Defaults to 0.
00504         @param K_factor (float, optional): Rigidity factor. Defaults to 3 (assuming cantilever).
00505         @param L_0 (float, optional): Position of the inflection point.
00506             Defaults to -1, e.g. computed as the total length, assuming cantilever.
00507         @param L_b (float, optional):Maximal unbraced lateral torsional buckling length.
00508             Defaults to -1, e.g computed as the total length, assuming cantilever with no bracing support.
00509         @param Mc (float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
00510         @param K (float, optional): Residual strength ratio. Defaults to -1, e.g. computed in ComputeK.
00511         @param theta_u (float, optional): Ultimate rotation. Defaults to -1, e.g. computed in ComputeTheta_u.
00512         @param safety_factors (bool, optional): Safety factors used if standard mechanical parameters are used (not test results). Defaults to False.
00513         """
00514         self.section = deepcopy(section)
00515         super().__init__(ID, section.Type, section.d, section.bf, section.tf, section.tw, section.h_1,
00516             section.Iy_mod, section.iz, section.E, section.Fy, section.Npl, section.My, section.L, N_G,
00517             K_factor, L_0, L_b, Mc, K, theta_u, safety_factors)
00518         self.section_name_tag = section.name_tag
00519         self.UpdateStoredData()
00520 
00521 
```

7.35.3 Member Data Documentation

7.35.3.1 section

`section`

Definition at line 514 of file MaterialModels.py.

7.35.3.2 section_name_tag

`section_name_tag`

Definition at line 518 of file MaterialModels.py.

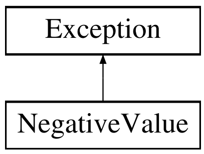
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.36 NegativeValue Class Reference

Exception class for the "negative value (argument or result)" error.

Inheritance diagram for NegativeValue:



7.36.1 Detailed Description

Exception class for the "negative value (argument or result)" error.

Definition at line 16 of file ErrorHandling.py.

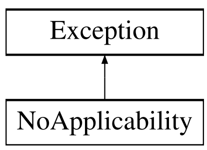
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py

7.37 NoApplicability Class Reference

Exception class for the "no applicability of formula of theory" error.

Inheritance diagram for NoApplicability:



7.37.1 Detailed Description

Exception class for the "no applicability of formula of theory" error.

Definition at line 47 of file ErrorHandling.py.

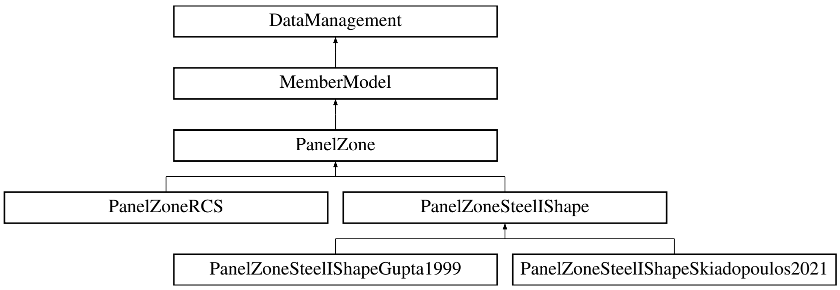
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py

7.38 PanelZone Class Reference

Class that handles the storage and manipulation of a panel zone's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Inheritance diagram for PanelZone:



Public Member Functions

- def __init__ (self, int master_node_ID, mid_panel_zone_width, mid_panel_zone_height, E, A_rigid, I_rigid, int geo_transf_ID, int mat_ID, pin_corners=True)

    Constructor of the class.
- def CreateMember (self)

    Method that initialises the member by calling the OpenSeesPy commands through various functions.
- def Record (self, str name_txt, str data_dir, force_rec=True, def_rec=True, time_rec=True)

    Implementation of the homonym abstract method.
- def RecordNodeDef (self, str name_txt, str data_dir, time_rec=True)

    Implementation of the homonym abstract method.
- def ReInit (self)

    Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False)

    Implementation of the homonym abstract method.
- def UpdateStoredData (self)

    Implementation of the homonym abstract method.

Public Attributes

- A_rigid
- beam_section_name_tag
- col_section_name_tag
- data
- E
- element_array
- geo_transf_ID
- I_rigid
- Initialized
- iNode_ID
- jNode_ID
- master_node_ID
- mat_ID
- mid_panel_zone_height
- mid_panel_zone_width
- pin_corners
- spring_ID

7.38.1 Detailed Description

Class that handles the storage and manipulation of a panel zone's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Parameters

MemberModel	Parent abstract class.

Definition at line 94 of file MemberModel.py.

7.38.2 Constructor & Destructor Documentation

7.38.2.1 __init__()

```
def __init__ (
              self,
             int master_node_ID,
              mid_panel_zone_width,
              mid_panel_zone_height,
              E,
              A_rigid,
              I_rigid,
             int geo_transf_ID,
             int mat_ID,
              pin_corners = True )
```

Constructor of the class.

Parameters

master_node_ID	(int): ID of the master node (central top node that should be a grid node).
mid_panel_zone_width	(float): Mid panel zone width.
mid_panel_zone_height	(float): Mid panel zone height.
E	(float): Young modulus.
A_rigid	(float): A very rigid area.
I_rigid	(float): A very rigid moment of inertia.
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
mat_ID	(int): ID of the material model for the panel zone spring.
pin_corners	(bool, optional): Option to pin the corners (xy03/xy04, xy06/xy07, xy09/xy10) or not. Used for RCS models. Defaults to True.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	mid_panel_zone_width needs to be positive.
NegativeValue	mid_panel_zone_height needs to be positive.
NegativeValue	E needs to be positive.
NegativeValue	A_rigid needs to be positive.
NegativeValue	I_rigid needs to be positive.
NegativeValue	geo_tranf_ID needs to be a positive integer.
NegativeValue	mat_ID needs to be a positive integer.

Reimplemented in PanelZoneRCS, PanelZoneSteelIShape, PanelZoneSteelIShapeGupta1999, and PanelZoneSteelIShapeSkiadopo

Definition at line 100 of file MemberModel.py.

```
00100     def __init__(self, master_node_ID: int, mid_panel_zone_width, mid_panel_zone_height, E, A_rigid,
          I_rigid, geo_transf_ID: int, mat_ID: int, pin_corners = True):
00101         """
00102         Constructor of the class.
00103 
00104         @param master_node_ID (int): ID of the master node (central top node that should be a grid
          node).
00105         @param mid_panel_zone_width (float): Mid panel zone width.
00106         @param mid_panel_zone_height (float): Mid panel zone height.
00107         @param E (float): Young modulus.
00108         @param A_rigid (float): A very rigid area.
00109         @param I_rigid (float): A very rigid moment of inertia.
00110         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
          documentation).
00111         @param mat_ID (int): ID of the material model for the panel zone spring.
00112         @param pin_corners (bool, optional): Option to pin the corners (xy03/xy04, xy06/xy07,
          xy09/xy10) or not. Used for RCS models. Defaults to True.
00113 
00114         @exception NegativeValue: ID needs to be a positive integer.
00115         @exception NegativeValue: mid_panel_zone_width needs to be positive.
00116         @exception NegativeValue: mid_panel_zone_height needs to be positive.
00117         @exception NegativeValue: E needs to be positive.
00118         @exception NegativeValue: A_rigid needs to be positive.
00119         @exception NegativeValue: I_rigid needs to be positive.
00120         @exception NegativeValue: geo_tranf_ID needs to be a positive integer.
00121         @exception NegativeValue: mat_ID needs to be a positive integer.
00122         """
00123         # Check
00124         if master_node_ID < 1: raise NegativeValue()
00125         # if master_node_ID > 99: raise WrongNodeIDConvention(master_node_ID)
00126         if mid_panel_zone_width < 0: raise NegativeValue()
00127         if mid_panel_zone_height < 0: raise NegativeValue()
00128         if E < 0: raise NegativeValue()
00129         if A_rigid < 0: raise NegativeValue()
00130         if I_rigid < 0: raise NegativeValue()
00131         if geo_transf_ID > 1: raise NegativeValue()
00132         if mat_ID < 0: raise NegativeValue()
```

```
        # Arguments
        self.master_node_ID = master_node_ID
        self.mid_panel_zone_width = mid_panel_zone_width
        self.mid_panel_zone_height = mid_panel_zone_height
        self.E = E
        self.A_rigid = A_rigid
        self.I_rigid = I_rigid
        self.geo_transf_ID = geo_transf_ID
        self.mat_ID = mat_ID
        self.pin_corners = pin_corners

        # Initialized the parameters that are dependent from others
        self.col_section_name_tag = "None"
        self.beam_section_name_tag = "None"
        self.Initialized = False
        self.ReInit()
```

7.38.3 Member Function Documentation

7.38.3.1 CreateMember()

def CreateMember (
self)

Method that initialises the member by calling the OpenSeesPy commands through various functions.

Definition at line 220 of file MemberModel.py.

```
00220     def CreateMember(self):
00221         """
00222         Method that initialises the member by calling the OpenSeesPy commands through various functions.
00223         """
00224         # Define nodes
00225         DefinePanelZoneNodes(self.master_node_ID, self.mid_panel_zone_width, self.mid_panel_zone_height)
00226         xy1 = IDConvention(self.master_node_ID, 1)
00227         xy01 = IDConvention(self.master_node_ID, 1, 1)
00228         xy03 = IDConvention(self.master_node_ID, 3, 1)
00229         xy04 = IDConvention(self.master_node_ID, 4, 1)
00230         xy06 = IDConvention(self.master_node_ID, 6, 1)
00231         xy07 = IDConvention(self.master_node_ID, 7, 1)
00232         xy09 = IDConvention(self.master_node_ID, 9, 1)
00233         xy10 = IDConvention(self.master_node_ID, 10)
00234
00235         # Define rigid elements
00236         self.element_array = DefinePanelZoneElements(self.master_node_ID, self.E, self.A_rigid, self.I_rigid, self.geo_transf_ID)
00237
00238         # Define zero length element
00239         self.spring_ID = IDConvention(xy1, xy01)
00240         RotationalSpring(self.spring_ID, xy1, xy01, self.mat_ID)
00241         self.element_array.append([self.spring_ID, xy1, xy01])
00242         self.iNode_ID = xy1
00243         self.jNode_ID = xy01
00244
00245         # Pin connections
00246         if self.pin_corners:
00247             Pin(xy03, xy04)
00248             Pin(xy06, xy07)
00249             Pin(xy09, xy10)
00250
00251         # Update class
00252         self.Initialized = True
00253         self.UpdateStoredData()
00254
00255
```

7.38.3.2 Record()

```
def Record (
              self,
             str name_txt,
             str data_dir,
              force_rec = True,
              def_rec = True,
              time_rec = True )
```

Implementation of the homonym abstract method.

See parent class MemberModel for detailed information.

Reimplemented from MemberModel.

Definition at line 256 of file MemberModel.py.

```
00256     def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
00257         """
00258         Implementation of the homonym abstract method.
00259         See parent class MemberModel for detailed information.
00260         """
00261         super().Record(self.spring_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec,
      time_rec=time_rec)
00262
00263
```

7.38.3.3 RecordNodeDef()

```
def RecordNodeDef (
              self,
             str name_txt,
             str data_dir,
              time_rec = True )
```

Implementation of the homonym abstract method.

See parent class MemberModel for detailed information.

Reimplemented from MemberModel.

Definition at line 264 of file MemberModel.py.

```
00264     def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
00265         """
00266         Implementation of the homonym abstract method.
00267         See parent class MemberModel for detailed information.
00268         """
00269         super().RecordNodeDef(self.iNode_ID, self.jNode_ID, name_txt, data_dir, time_rec=time_rec)
00270
00271
```

7.38.3.4 ReInit()

```
def ReInit (
              self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 152 of file MemberModel.py.

```
    def ReInit(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        # Arguments
        self.spring_ID = -1

        # Members
        if self.col_section_name_tag != "None": self.col_section_name_tag = self.col_section_name_tag
      + " (modified)"
        if self.beam_section_name_tag != "None": self.beam_section_name_tag =
      self.beam_section_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredData()


```

7.38.3.5 ShowInfo()

```
def ShowInfo (
              self,
              plot = False,
              block = False )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the material model. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 189 of file MemberModel.py.

```
    def ShowInfo(self, plot = False, block = False):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to
      False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
      of the program everytime that a plot should pop up). Defaults to False.
        """
        print("")
        print("Requested info for Panel Zone member model, master node ID =
      {}".format(self.master_node_ID))
        print("Section associated, column: {} ".format(self.col_section_name_tag))
        print("Section associated, beam: {} ".format(self.beam_section_name_tag))
        print("Material model of the panel zone ID = {}".format(self.mat_ID))
```

```
        print("Spring ID = {} (if -1, not defined yet)".format(self.spring_ID))
        print("Mid panel zone width = {} mm".format(self.mid_panel_zone_width/mm_unit))
        print("Mid panel zone height = {} mm".format(self.mid_panel_zone_height/mm_unit))
        print("Young modulus E = {} GPa".format(self.E/GPa_unit))
        print("Area of the elements (rigid) = {} mm2".format(self.A_rigid/mm2_unit))
        print("Moment of inetia of the elements (strong axis, rigid) = {} mm4".format(self.I_rigid/mm4_unit))
        print("Geometric transformation = {}".format(self.geo_transf_ID))
        print("")

        if plot:
            if self.Initialized:
                plot_member(self.element_array, "Panel zone, ID = {}".format(self.master_node_ID))
                if block:
                    plt.show()
            else:
                print("The panel zone is not initialized (node and elements not created) for master node ID = {}".format(self.master_node_ID))


```

7.38.3.6 UpdateStoredData()

def UpdateStoredData (
    self)

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 169 of file MemberModel.py.

```
00169    def UpdateStoredData(self):
00170        """
00171        Implementation of the homonym abstract method.
00172        See parent class DataManagement for detailed information.
00173        """
00174        self.data = [["INFO_TYPE", "PanelZone"], # Tag for differentiating different data
00175            ["master_node_ID", self.master_node_ID],
00176            ["col_section_name_tag", self.col_section_name_tag],
00177            ["beam_section_name_tag", self.beam_section_name_tag],
00178            ["mat_ID", self.mat_ID],
00179            ["spring_ID", self.spring_ID],
00180            ["mid_panel_zone_width", self.mid_panel_zone_width],
00181            ["mid_panel_zone_height", self.mid_panel_zone_height],
00182            ["E", self.E],
00183            ["A_rigid", self.A_rigid],
00184            ["I_rigid", self.I_rigid],
00185            ["tranf_ID", self.geo_transf_ID],
00186            ["Initialized", self.Initialized]]
00187
00188
```

7.38.4 Member Data Documentation

7.38.4.1 A_rigid

A_rigid

Definition at line 139 of file MemberModel.py.

7.38.4.2 beam_section_name_tag

```
beam_section_name_tag
```

Definition at line 147 of file MemberModel.py.

7.38.4.3 col_section_name_tag

```
col_section_name_tag
```

Definition at line 146 of file MemberModel.py.

7.38.4.4 data

```
data
```

Definition at line 174 of file MemberModel.py.

7.38.4.5 E

```
E
```

Definition at line 138 of file MemberModel.py.

7.38.4.6 element_array

```
element_array
```

Definition at line 236 of file MemberModel.py.

7.38.4.7 geo_transf_ID

```
geo_transf_ID
```

Definition at line 141 of file MemberModel.py.

7.38.4.8 I_rigid

```
I_rigid
```

Definition at line 140 of file MemberModel.py.

7.38.4.9 Initialized

```
Initialized
```

Definition at line 148 of file MemberModel.py.

7.38.4.10 iNode_ID

```
iNode_ID
```

Definition at line 242 of file MemberModel.py.

7.38.4.11 jNode_ID

```
jNode_ID
```

Definition at line 243 of file MemberModel.py.

7.38.4.12 master_node_ID

```
master_node_ID
```

Definition at line 135 of file MemberModel.py.

7.38.4.13 mat_ID

```
mat_ID
```

Definition at line 142 of file MemberModel.py.

7.38.4.14 mid_panel_zone_height

`mid_panel_zone_height`

Definition at line 137 of file MemberModel.py.

7.38.4.15 mid_panel_zone_width

`mid_panel_zone_width`

Definition at line 136 of file MemberModel.py.

7.38.4.16 pin_corners

`pin_corners`

Definition at line 143 of file MemberModel.py.

7.38.4.17 spring_ID

`spring_ID`

Definition at line 158 of file MemberModel.py.

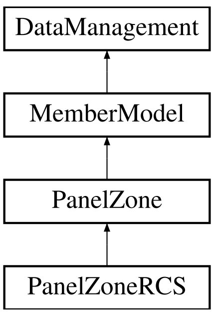
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.39 PanelZoneRCS Class Reference

WIP: Class that is the children of PanelZone and it's used for the panel zone in a RCS (RC column continous, Steel beam).

Inheritance diagram for PanelZoneRCS:



Public Member Functions

- def __init__ (self, int master_node_ID, RCRectShape col, SteelIShape beam, int geo_transf_ID, int mat_ID, rigid=RIGID)

    Constructor of the class.

Public Attributes

- beam
- beam_section_name_tag
- col
- col_section_name_tag

7.39.1 Detailed Description

WIP: Class that is the children of PanelZone and it's used for the panel zone in a RCS (RC column continous, Steel beam).

Note that the corners are not pinned (do it manually).

Parameters

PanelZone	Parent class.

Definition at line 306 of file MemberModel.py.

7.39.2 Constructor & Destructor Documentation

7.39.2.1 __init__()

```
def __init__ (
                self,
                int master_node_ID,
                RCRectShape col,
                SteelIShape beam,
                int geo_transf_ID,
                int mat_ID,
                rigid = RIGID )
```

Constructor of the class.

Parameters

master_node_ID	(int): ID of the master node (central top node that should be a grid node).
col	(RCRectShape): RCRectShape column section object.
beam	(SteelIShape): SteelIShape beam section object.
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
mat_ID	(int): ID of the material model for the panel zone spring.
rigid	(float, optional): Parameter with a value enough big to assure rigidity of one element but enough small to avoid convergence problem. Defaults to RIGID.

Reimplemented from PanelZone.

Definition at line 313 of file MemberModel.py.

```
00313     def __init__(self, master_node_ID: int, col: RCRectShape, beam: SteelIShape, geo_transf_ID: int,
      mat_ID: int, rigid = RIGID):
00314         """
00315         Constructor of the class.
00316 
00317         @param master_node_ID (int): ID of the master node (central top node that should be a grid
      node).
00318         @param col (RCRectShape): RCRectShape column section object.
00319         @param beam (SteelIShape): SteelIShape beam section object.
00320         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
      documentation).
00321         @param mat_ID (int): ID of the material model for the panel zone spring.
00322         @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one
      element
00323             but enough small to avoid convergence problem. Defaults to RIGID.
00324         """
00325         self.col = deepcopy(col)
00326         self.beam = deepcopy(beam)
00327         super().__init__(master_node_ID, col.d/2.0, beam.d/2.0, beam.E, max(col.A, beam.A)*rigid,
      max(col.Iy, beam.Iy)*rigid, geo_transf_ID, mat_ID, False)
00328 
00329         self.col_section_name_tag = col.name_tag
00330         self.beam_section_name_tag = beam.name_tag
00331         self.UpdateStoredData()
00332 
00333 
```

7.39.3 Member Data Documentation

7.39.3.1 beam

```
beam
```

Definition at line 326 of file MemberModel.py.

7.39.3.2 beam_section_name_tag

```
beam_section_name_tag
```

Definition at line 330 of file MemberModel.py.

7.39.3.3 col

```
col
```

Definition at line 325 of file MemberModel.py.

7.39.3.4 col_section_name_tag

`col_section_name_tag`

Definition at line 329 of file MemberModel.py.

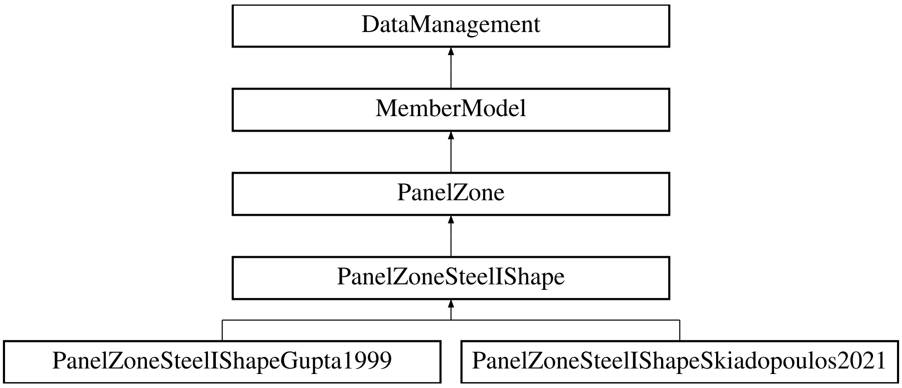
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.40 PanelZoneSteelIShape Class Reference

Class that is the children of PanelZone and combine the class SteelIShape (section) to retrieve the information needed.

Inheritance diagram for PanelZoneSteelIShape:



Public Member Functions

- def __init__ (self, int master_node_ID, SteelIShape col, SteelIShape beam, int geo_transf_ID, int mat_ID, rigid=RIGID)

    Constructor of the class.

Public Attributes

- beam
- beam_section_name_tag
- col
- col_section_name_tag

7.40.1 Detailed Description

Class that is the children of PanelZone and combine the class SteelIShape (section) to retrieve the information needed.

Parameters

PanelZone	Parent class.

Definition at line 279 of file MemberModel.py.

7.40.2 Constructor & Destructor Documentation

7.40.2.1 __init__()

```
def __init__ (
              self,
             int master_node_ID,
             SteelIShape col,
             SteelIShape beam,
             int geo_transf_ID,
             int mat_ID,
              rigid = RIGID )
```

Constructor of the class.

Parameters

master_node_ID	(int): ID of the master node (central top node that should be a grid node).
col	(SteelIShape): SteelIShape column section object.
beam	(SteelIShape): SteelIShape beam section object.
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
mat_ID	(int): ID of the material model for the panel zone spring.
rigid	(float, optional): Parameter with a value enough big to assure rigidity of one element but enough small to avoid convergence problem. Defaults to RIGID.

Reimplemented from PanelZone.

Reimplemented in PanelZoneSteelIShapeGupta1999, and PanelZoneSteelIShapeSkiadopoulos2021.

Definition at line 285 of file MemberModel.py.

```
00285     def __init__(self, master_node_ID: int, col: SteelIShape, beam: SteelIShape, geo_transf_ID: int, mat_ID: int, rigid = RIGID):
00286         """
00287         Constructor of the class.
00288 
00289         @param master_node_ID (int): ID of the master node (central top node that should be a grid node).
00290         @param col (SteelIShape): SteelIShape column section object.
00291         @param beam (SteelIShape): SteelIShape beam section object.
00292         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
00293         @param mat_ID (int): ID of the material model for the panel zone spring.
00294         @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one element
00295             but enough small to avoid convergence problem. Defaults to RIGID.
00296         """
00297         self.col = deepcopy(col)
00298         self.beam = deepcopy(beam)
```

```
00299         super().__init__(master_node_ID, col.d/2.0, beam.d/2.0, col.E, max(col.A, beam.A)*rigid,
     max(col.Iy, beam.Iy)*rigid, geo_transf_ID, mat_ID)
00300
00301         self.col_section_name_tag = col.name_tag
00302         self.beam_section_name_tag = beam.name_tag
00303         self.UpdateStoredData()
00304
00305
```

7.40.3 Member Data Documentation

7.40.3.1 beam

`beam`

Definition at line 298 of file MemberModel.py.

7.40.3.2 beam_section_name_tag

`beam_section_name_tag`

Definition at line 302 of file MemberModel.py.

7.40.3.3 col

`col`

Definition at line 297 of file MemberModel.py.

7.40.3.4 col_section_name_tag

`col_section_name_tag`

Definition at line 301 of file MemberModel.py.

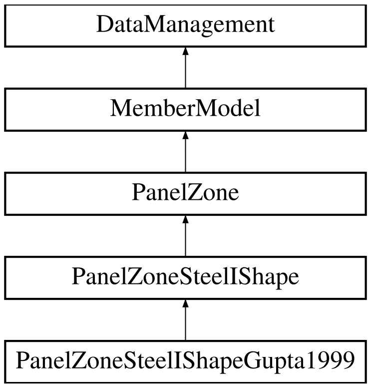
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.41 PanelZoneSteelIShapeGupta1999 Class Reference

Class that is the children of PanelZoneSteelIShape and automatically create the spring material model Gupta 1999 (ID = master_node_ID).

Inheritance diagram for PanelZoneSteelIShapeGupta1999:



Public Member Functions

- def __init__ (self, int master_node_ID, SteelIShape col, SteelIShape beam, int geo_transf_ID, t_dp=0, rigid=RIGID)

    Constructor of the class.

Public Attributes

- beam
- col

7.41.1 Detailed Description

Class that is the children of PanelZoneSteelIShape and automatically create the spring material model Gupta 1999 (ID = master_node_ID).

Parameters

PanelZoneSteelIShape	Parent class.

Definition at line 334 of file MemberModel.py.

7.41.2 Constructor & Destructor Documentation

7.41.2.1 __init__()

```
def __init__ (
                self,
               int master_node_ID,
               SteelIShape col,
               SteelIShape beam,
               int geo_transf_ID,
                t_dp = 0,
                rigid = RIGID )
```

Constructor of the class.

Parameters

master_node_ID	(int): ID of the master node (central top node that should be a grid node).
col	(SteelIShape): SteelIShape column section object.
beam	(SteelIShape): SteelIShape beam section object.
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
t_dp	(float, optional): Doubler plate thickness. Defaults to 0.
rigid	(float, optional): Parameter with a value enough big to assure rigidity of one element but enough small to avoid convergence problem. Defaults to RIGID.

Reimplemented from PanelZoneSteelIShape.

Definition at line 340 of file MemberModel.py.

```
00340     def __init__(self, master_node_ID: int, col: SteelIShape, beam: SteelIShape, geo_transf_ID: int,
       t_dp = 0, rigid=RIGID):
00341         """
00342         Constructor of the class.
00343 
00344         @param master_node_ID (int): ID of the master node (central top node that should be a grid
       node).
00345         @param col (SteelIShape): SteelIShape column section object.
00346         @param beam (SteelIShape): SteelIShape beam section object.
00347         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
       documentation).
00348         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.
00349         @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one
       element
00350             but enough small to avoid convergence problem. Defaults to RIGID.
00351         """
00352         self.col = deepcopy(col)
00353         self.beam = deepcopy(beam)
00354         mat_ID = master_node_ID
00355         pz_spring = Gupta1999SteelIShape(mat_ID, col, beam, t_dp)
00356         pz_spring.Hysteretic()
00357 
00358         super().__init__(master_node_ID, col, beam, geo_transf_ID, mat_ID, rigid)
00359 
00360 
```

7.41.3 Member Data Documentation

7.41.3.1 beam

```
beam
```

Definition at line 353 of file MemberModel.py.

7.41.3.2 col

```
col
```

Definition at line 352 of file MemberModel.py.

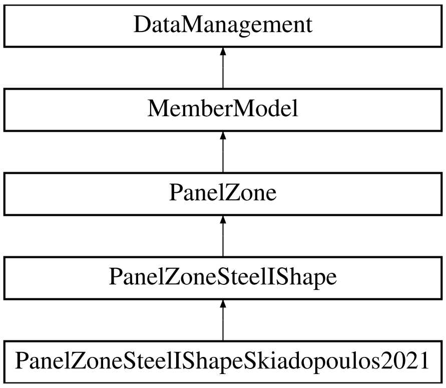
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.42 PanelZoneSteelIShapeSkiadopoulos2021 Class Reference

Class that is the children of PanelZoneSteelIShape and automatically create the spring material model Skiadopoulos 2021 (ID = master_node_ID).

Inheritance diagram for PanelZoneSteelIShapeSkiadopoulos2021:



Public Member Functions

- def __init__ (self, int master_node_ID, SteelIShape col, SteelIShape beam, int geo_transf_ID, t_dp=0, rigid=RIGID)

    Constructor of the class.

Public Attributes

- beam
- col

7.42.1 Detailed Description

Class that is the children of PanelZoneSteelIShape and automatically create the spring material model Skiadopoulos 2021 (ID = master_node_ID).

Parameters

PanelZoneSteelIShape	Parent class.

Definition at line 361 of file MemberModel.py.

7.42.2 Constructor & Destructor Documentation

7.42.2.1 __init__()

```
def __init__ (
             self,
            int master_node_ID,
            SteelIShape col,
            SteelIShape beam,
            int geo_transf_ID,
             t_dp = 0,
             rigid = RIGID )
```

Constructor of the class.

Parameters

master_node_ID	(int): ID of the master node (central top node that should be a grid node).
col	(SteelIShape): SteelIShape column section object.
beam	(SteelIShape): SteelIShape beam section object.
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
t_dp	(float, optional): Doubler plate thickness. Defaults to 0.
rigid	(float, optional): Parameter with a value enough big to assure rigidity of one element but enough small to avoid convergence problem. Defaults to RIGID.

Reimplemented from PanelZoneSteelIShape.

Definition at line 367 of file MemberModel.py.

```
00367     def __init__(self, master_node_ID: int, col: SteelIShape, beam: SteelIShape, geo_transf_ID: int,
      t_dp = 0, rigid=RIGID):
00368         """
00369         Constructor of the class.
00370 
00371         @param master_node_ID (int): ID of the master node (central top node that should be a grid
      node).
00372         @param col (SteelIShape): SteelIShape column section object.
00373         @param beam (SteelIShape): SteelIShape beam section object.
00374         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
      documentation).
00375         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.
00376         @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one
      element
00377             but enough small to avoid convergence problem. Defaults to RIGID.
00378         """
00379         self.col = deepcopy(col)
00380         self.beam = deepcopy(beam)
00381         mat_ID = master_node_ID
00382         pz_spring = Skiadopoulos2021SteelIShape(mat_ID, col, beam, t_dp)
00383         pz_spring.Hysteretic()
00384 
```

```
        super().__init__(master_node_ID, col, beam, geo_transf_ID, mat_ID, rigid)
```

7.42.3 Member Data Documentation

7.42.3.1 beam

`beam`

Definition at line 380 of file MemberModel.py.

7.42.3.2 col

`col`

Definition at line 379 of file MemberModel.py.

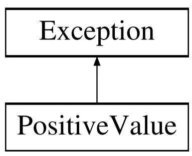
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.43 PositiveValue Class Reference

Exception class for the "positive value (argument or result)" error.

Inheritance diagram for PositiveValue:



7.43.1 Detailed Description

Exception class for the "positive value (argument or result)" error.

Definition at line 21 of file ErrorHandling.py.

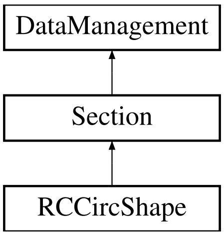
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py

7.44 RCCircShape Class Reference

Class that stores funcions, geometric and mechanical properties of RC circular shape profile.

Inheritance diagram for RCCircShape:



Public Member Functions

- def __init__ (self, b, L, e, fc, D_bars, int n_bars, fy, Ey, D_hoops, s, fs, Es, name_tag="Not Defined", rho_s_vol=-1, Ec=-1)

    The conctructor of the class.
- def ComputeEc (self)

    Compute Ec using the formula from Mander et Al.
- def ComputeI (self)

    Compute the moment of inertia of the circular section.
- def ComputeRhoVol (self)

    Compute the ratio of the volume of transverse confining steel to the volume of confined concrete core.
- def ReInit (self, rho_s_vol=-1, Ec=-1)

    Implementation of the homonym abstract method.
- def ShowInfo (self)

    Implementation of the homonym abstract method.
- def UpdateStoredData (self)

    Implementation of the homonym abstract method.

Public Attributes

- A
- Ac
- As
- Ay
- b
- bc
- cl_bars
- cl_hoops
- D_bars
- D_hoops
- data
- e
- Ec
- Es
- Ey
- fc

- fs
- fy
- l
- L
- n_bars
- name_tag
- rho_bars
- rho_s_vol
- s

7.44.1 Detailed Description

Class that stores funcions, geometric and mechanical properties of RC circular shape profile.

Note that for the validity of the formulas, the hoops needs to be closed (with 135 degress possibly).

Parameters

Section	Parent abstract class.

Definition at line 570 of file Section.py.

7.44.2 Constructor & Destructor Documentation

7.44.2.1 __init__()

```
def __init__ (
              self,
              b,
              L,
              e,
              fc,
              D_bars,
             int n_bars,
              fy,
              Ey,
              D_hoops,
              s,
              fs,
              Es,
              name_tag = "Not Defined",
              rho_s_vol = -1,
              Ec = -1 )
```

The conctructor of the class.

Parameters

b	(float): Width of the section.

Parameters

L	(float): Effective length of the element associated with this section. If the panel zone is present, exclude its dimension.
e	(float): Concrete cover.
fc	(float): Unconfined concrete compressive strength (cylinder test).
D_bars	(float): Diameter of the vertical reinforcing bars.
n_bars	(int): Number of vertical reinforcing bars.
fy	(float): Yield stress for reinforcing bars.
Ey	(float): Young modulus for reinforcing bars.
D_hoops	(float): Diameter of the hoops.
s	(float): Vertical centerline spacing between hoops.
fs	(float): Yield stress for the hoops.
Es	(float): Young modulus for the hoops
name_tag	(str, optional): A nametag for the section. Defaults to "Not Defined".
rho_s_vol	(float, optional): Ratio of the volume of transverse confining steel to the volume of confined concrete core. Defaults to -1, e.g. computed according to Mander et Al. 1988.
Ec	(float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in **init**() and ReInit().

Exceptions

NegativeValue	b needs to be positive.
NegativeValue	L needs to be positive.
NegativeValue	e needs to be positive.
PositiveValue	fc needs to be negative.
NegativeValue	D_bars needs to be positive.
NegativeValue	n_bars needs to be a positive integer.
NegativeValue	fy needs to be positive.
NegativeValue	Ey needs to be positive.
NegativeValue	D_hoops needs to be positive.
NegativeValue	s needs to be positive.
NegativeValue	fs needs to be positive.
NegativeValue	Es needs to be positive.
NegativeValue	Ec needs to be positive if different from -1.
InconsistentGeometry	e should be smaller than half the depth and the width of the section.

Definition at line 577 of file Section.py.

```
00577     def __init__(self, b, L, e, fc, D_bars, n_bars: int, fy, Ey, D_hoops, s, fs, Es, name_tag = "Not
      Defined", rho_s_vol = -1, Ec = -1):
00578         """
00579         The conctructor of the class.
00580 
00581         @param b (float): Width of the section.
00582         @param L (float): Effective length of the element associated with this section.
00583             If the panel zone is present, exclude its dimension.
00584         @param e (float): Concrete cover.
00585         @param fc (float): Unconfined concrete compressive strength (cylinder test).
00586         @param D_bars (float): Diameter of the vertical reinforcing bars.
00587         @param n_bars (int): Number of vertical reinforcing bars.
00588         @param fy (float): Yield stress for reinforcing bars.
00589         @param Ey (float): Young modulus for reinforcing bars.
00590         @param D_hoops (float): Diameter of the hoops.
00591         @param s (float): Vertical centerline spacing between hoops.
00592         @param fs (float): Yield stress for the hoops.
00593         @param Es (float): Young modulus for the hoops
00594         @param name_tag (str, optional): A nametag for the section. Defaults to "Not Defined".
00595         @param rho_s_vol (float, optional): Ratio of the volume of transverse confining steel to the
      volume of confined concrete core.
```

```
            Defaults to -1, e.g. computed according to Mander et Al. 1988.
        @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in __init__() and ReInit().

        @exception NegativeValue: b needs to be positive.
        @exception NegativeValue: L needs to be positive.
        @exception NegativeValue: e needs to be positive.
        @exception PositiveValue: fc needs to be negative.
        @exception NegativeValue: D_bars needs to be positive.
        @exception NegativeValue: n_bars needs to be a positive integer.
        @exception NegativeValue: fy needs to be positive.
        @exception NegativeValue: Ey needs to be positive.
        @exception NegativeValue: D_hoops needs to be positive.
        @exception NegativeValue: s needs to be positive.
        @exception NegativeValue: fs needs to be positive.
        @exception NegativeValue: Es needs to be positive.
        @exception NegativeValue: Ec needs to be positive if different from -1.
        @exception InconsistentGeometry: e should be smaller than half the depth and the width of the section.
        """
        # Check
        if b < 0: raise NegativeValue()
        if L < 0: raise NegativeValue()
        if e < 0: raise NegativeValue()
        if fc > 0: raise PositiveValue()
        if D_bars < 0: raise NegativeValue()
        if n_bars < 0: raise NegativeValue()
        if fy < 0: raise NegativeValue()
        if Ey < 0: raise NegativeValue()
        if D_hoops < 0: raise NegativeValue()
        if s < 0: raise NegativeValue()
        if fs < 0: raise NegativeValue()
        if Es < 0: raise NegativeValue()
        if Ec != -1 and Ec < 0: raise NegativeValue()
        if e > b/2: raise InconsistentGeometry()

        # Arguments
        self.b = b
        self.L = L
        self.e = e
        self.fc = fc
        self.D_bars = D_bars
        self.n_bars = n_bars
        self.fy = fy
        self.Ey = Ey
        self.D_hoops = D_hoops
        self.s = s
        self.fs = fs
        self.Es = Es
        self.name_tag = name_tag

        # Initialized the parameters that are dependent from others
        self.ReInit(rho_s_vol, Ec)


```

7.44.3 Member Function Documentation

7.44.3.1 ComputeEc()

```
def ComputeEc (
                self )
```

Compute Ec using the formula from Mander et Al.

1988.

Returns

float: Young modulus of concrete.

Definition at line 744 of file Section.py.

```
00744     def ComputeEc(self):
00745         """
00746         Compute Ec using the formula from Mander et Al. 1988.
00747 
00748         @returns float: Young modulus of concrete.
00749         """
00750 
00751         return 5000.0 * math.sqrt(-self.fc/MPa_unit) * MPa_unit
00752 
00753 
```

7.44.3.2 ComputeI()

```
def ComputeI (
              self )
```

Compute the moment of inertia of the circular section.

Returns

float: Moment of inertia.

Definition at line 754 of file Section.py.

```
00754     def ComputeI(self):
00755         """
00756         Compute the moment of inertia of the circular section.
00757 
00758         @returns float: Moment of inertia.
00759         """
00760         return self.b**4*math.pi/64
00761 
00762 
```

7.44.3.3 ComputeRhoVol()

```
def ComputeRhoVol (
              self )
```

Compute the ratio of the volume of transverse confining steel to the volume of confined concrete core.

(according to Mander et Al. 1988).

Returns

float: Ratio.

Definition at line 731 of file Section.py.

```
00731     def ComputeRhoVol(self):
00732         """
00733         Compute the ratio of the volume of transverse confining steel to the volume of confined
        concrete core.
00734         (according to Mander et Al. 1988).
00735 
00736         @returns float: Ratio.
00737         """
00738         vol_s = self.As*math.pi*self.bc
00739         vol_c = math.pi/4*self.bc**2*self.s
00740 
00741         return vol_s/vol_c
00742 
00743 
```

7.44.3.4 ReInit()

```
def ReInit (
        self,
        rho_s_vol = -1,
        Ec = -1 )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

rho_s_vol	(float, optional): Ratio of the volume of transverse confining steel to the volume of confined concrete core. Defaults to -1, e.g. computed according to Mander et Al. 1988.
Ec	(float): Young modulus for concrete. Defaults to -1, e.g. computed according to Mander et Al. 1988.

Definition at line 649 of file Section.py.

```
00649     def ReInit(self, rho_s_vol = -1, Ec = -1):
00650         """
00651         Implementation of the homonym abstract method.
00652         See parent class DataManagement for detailed information.
00653 
00654         @param rho_s_vol (float, optional): Ratio of the volume of transverse confining steel to the volume of confined concrete core.
00655             Defaults to -1, e.g. computed according to Mander et Al. 1988.
00656         @param Ec (float): Young modulus for concrete. Defaults to -1, e.g. computed according to Mander et Al. 1988.
00657         """
00658         # Precompute some members
00659         self.cl_hoops = self.e + self.D_hoops/2.0 # centerline distance from the border of the extreme confining hoops
00660         self.cl_bars = self.e + self.D_bars/2.0 + self.D_hoops # centerline distance from the border of the corner bars
00661         self.bc = self.b - self.cl_hoops*2 # diameter of spiral (hoops) between bar centerline
00662         self.As = ComputeACircle(self.D_hoops)
00663 
00664         # Arguments
00665         self.rho_s_vol = self.ComputeRhoVol() if rho_s_vol == -1 else rho_s_vol
00666         self.Ec = self.ComputeEc() if Ec == -1 else Ec
00667 
00668         # Members
00669         self.A = ComputeACircle(self.b)
00670         self.Ac = ComputeACircle(self.bc)
00671         self.Ay = ComputeACircle(self.D_bars)
00672         self.rho_bars = ComputeRho(self.Ay, self.n_bars, self.A)
00673         self.I = self.ComputeI()
00674 
00675         # Data storage for loading/saving
00676         self.UpdateStoredData()
00677 
00678 
```

7.44.3.5 ShowInfo()

```
def ShowInfo (
        self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 710 of file Section.py.

```
    def ShowInfo(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        print("")
        print("Requested info for RC circular section of name tag = {}".format(self.name_tag))
        print("Width of the section b = {} mm".format(self.b/mm_unit))
        print("Concrete cover e = {} mm".format(self.e/mm_unit))
        print("Concrete area A = {} mm2".format(self.A/mm2_unit))
        print("Core concrete area Ac = {} mm2".format(self.Ac/mm2_unit))
        print("Unconfined concrete compressive strength fc = {} MPa".format(self.fc/MPa_unit))
        print("Young modulus for concrete Ec = {} GPa".format(self.Ec/GPa_unit))
        print("Diameter of the reinforcing bars D_bars = {} mm and area of one bar Ay = {} mm2 with {} bars".format(self.D_bars/mm_unit, self.Ay/mm2_unit, self.n_bars))
        print("Diameter of the hoops D_hoops = {} mm and area of one stirrup As = {} mm2".format(self.D_hoops/mm_unit, self.As/mm2_unit))
        print("Ratio of area of longitudinal reinforcement to area of concrete section rho_bars = {} ".format(self.rho_bars))
        print("Ratio of the volume of transverse confining steel to the volume of confined concrete core rho_s = {} ".format(self.rho_s_vol))
        print("Moment of inertia of the circular section I = {} mm4".format(self.I/mm4_unit))
        print("")


```

7.44.3.6 UpdateStoredData()

```
def UpdateStoredData (
            self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 679 of file Section.py.

```
00679     def UpdateStoredData(self):
00680         """
00681         Implementation of the homonym abstract method.
00682         See parent class DataManagement for detailed information.
00683         """
00684         self.data = [["INFO_TYPE", "RCCircShape"], # Tag for differentiating different data
00685             ["name_tag", self.name_tag],
00686             ["b", self.b],
00687             ["bc", self.bc],
00688             ["L", self.L],
00689             ["e", self.e],
00690             ["A", self.A],
00691             ["Ac", self.Ac],
00692             ["I", self.I],
00693             ["fc", self.fc],
00694             ["Ec", self.Ec],
00695             ["D_bars", self.D_bars],
00696             ["n_bars", self.n_bars],
00697             ["Ay", self.Ay],
00698             ["rho_bars", self.rho_bars],
00699             ["cl_bars", self.cl_bars],
00700             ["fy", self.fy],
00701             ["Ey", self.Ey],
00702             ["D_hoops", self.D_hoops],
00703             ["s", self.s],
00704             ["As", self.As],
00705             ["rho_s_vol", self.rho_s_vol],
00706             ["cl_hoops", self.cl_hoops],
00707             ["fs", self.fs],
00708             ["Es", self.Es]]
00709
```

7.44.4 Member Data Documentation

7.44.4.1 A

```
A
```

Definition at line 669 of file Section.py.

7.44.4.2 Ac

```
Ac
```

Definition at line 670 of file Section.py.

7.44.4.3 As

```
As
```

Definition at line 662 of file Section.py.

7.44.4.4 Ay

```
Ay
```

Definition at line 671 of file Section.py.

7.44.4.5 b

```
b
```

Definition at line 631 of file Section.py.

7.44.4.6 bc

```
bc
```

Definition at line 661 of file Section.py.

7.44.4.7 cl_bars

`cl_bars`

Definition at line 660 of file Section.py.

7.44.4.8 cl_hoops

`cl_hoops`

Definition at line 659 of file Section.py.

7.44.4.9 D_bars

`D_bars`

Definition at line 635 of file Section.py.

7.44.4.10 D_hoops

`D_hoops`

Definition at line 639 of file Section.py.

7.44.4.11 data

`data`

Definition at line 684 of file Section.py.

7.44.4.12 e

`e`

Definition at line 633 of file Section.py.

7.44.4.13 Ec

```
Ec
```

Definition at line 666 of file Section.py.

7.44.4.14 Es

```
Es
```

Definition at line 642 of file Section.py.

7.44.4.15 Ey

```
Ey
```

Definition at line 638 of file Section.py.

7.44.4.16 fc

```
fc
```

Definition at line 634 of file Section.py.

7.44.4.17 fs

```
fs
```

Definition at line 641 of file Section.py.

7.44.4.18 fy

```
fy
```

Definition at line 637 of file Section.py.

7.44.4.19 I

```
I
```

Definition at line 673 of file Section.py.

7.44.4.20 L

```
L
```

Definition at line 632 of file Section.py.

7.44.4.21 n_bars

```
n_bars
```

Definition at line 636 of file Section.py.

7.44.4.22 name_tag

```
name_tag
```

Definition at line 643 of file Section.py.

7.44.4.23 rho_bars

```
rho_bars
```

Definition at line 672 of file Section.py.

7.44.4.24 rho_s_vol

```
rho_s_vol
```

Definition at line 665 of file Section.py.

7.44.4.25 s

s

Definition at line 640 of file Section.py.

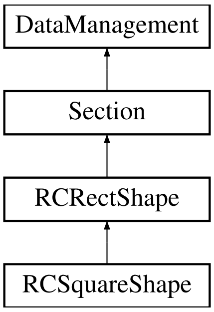
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Section.py

7.45 RCRectShape Class Reference

Class that stores funcions, geometric and mechanical properties of RC rectangular shape profile.

Inheritance diagram for RCRectShape:



Public Member Functions

- def __init__ (self, b, d, L, e, fc, D_bars, np.ndarray bars_position_x, np.ndarray bars_ranges_position_y, fy, Ey, D_hoops, s, fs, Es, name_tag="Not Defined", rho_s_x=-1, rho_s_y=-1, Ec=-1)

    The conctructor of the class.
- def ComputeA (self)

    Compute the area for a rectangular section.
- def ComputeAc (self)

    Compute the confined area (area inside the centerline of the hoops, according to Mander et Al.
- def ComputeEc (self)

    Compute Ec using the formula from Mander et Al.
- def ComputeIy (self)

    Compute the moment of inertia of the rectangular section with respect to the strong axis.
- def ComputeIz (self)

    Compute the moment of inertia of the rectangular section with respect to the weak axis.
- def ComputeNrBars (self)

    Compute the number of vertical bars in the array bars_position_x (note that this list of lists can have different list sizes).
- def ReInit (self, rho_s_x=-1, rho_s_y=-1, Ec=-1)

    Implementation of the homonym abstract method.
- def ShowInfo (self)

    Implementation of the homonym abstract method.
- def UpdateStoredData (self)

    Implementation of the homonym abstract method.

Public Attributes

- A
- Ac
- As
- Ay
- b
- bars_position_x
- bars_ranges_position_y
- bc
- cl_bars
- cl_hoops
- d
- D_bars
- D_hoops
- data
- dc
- e
- Ec
- Es
- Ey
- fc
- fs
- fy
- Iy
- Iz
- L
- name_tag
- nr_bars
- rho_bars
- rho_s_x
- rho_s_y
- s

7.45.1 Detailed Description

Class that stores funcions, geometric and mechanical properties of RC rectangular shape profile.

Note that for the validity of the formulas, at least one bar per corner and at least one hoop closed (with 135 degress possibly).

Parameters

Section	Parent abstract class.

Definition at line 264 of file Section.py.

7.45.2 Constructor & Destructor Documentation

7.45.2.1 __init__()

```
def __init__ (
              self,
              b,
              d,
              L,
              e,
              fc,
              D_bars,
             np.ndarray bars_position_x,
             np.ndarray bars_ranges_position_y,
              fy,
              Ey,
              D_hoops,
              s,
              fs,
              Es,
              name_tag = "Not Defined",
              rho_s_x = -1,
              rho_s_y = -1,
              Ec = -1 )
```

The conctructor of the class.

Parameters

b	(float): Width of the section.
d	(float): Depth of the section.
L	(float): Effective length of the element associated with this section. If the panel zone is present, exclude its dimension.
e	(float): Concrete cover.
fc	(float): Unconfined concrete compressive strength (cylinder test).
D_bars	(float): Diameter of the reinforcing bars.
bars_position_x	(np.ndarray): Array with a range of aligned vertical reinforcing bars for each row in x direction. Distances from border to bar centerline, bar to bar centerlines and finally bar centerline to border in the x direction (aligned). Starting from the left to right, from the top range to the bottom one. The number of bars for each range can vary; in this case, add this argument when defining the array " dtype = object".
bars_ranges_position↩_y	(np.ndarray): Array of dimension 1 with the position or spacing in y of the ranges in bars_position_x. Distances from border to range centerlines, range to range centerlines and finally range centerline to border in the y direction. Starting from the top range to the bottom one.
fy	(float): Yield stress for reinforcing bars.
Ey	(float): Young modulus for reinforcing bars.
D_hoops	(float): Diameter of the hoops.
s	(float): Centerline distance for the hoops.
fs	(float): Yield stress for the hoops.
Es	(float): Young modulus for the hoops
name_tag	(str, optional): A nametag for the section. Defaults to "Not Defined".
rho_s_x	(float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the x direction. Defaults to -1, e.g. computed in **init**() and ReInit() assuming one range of hoops.
rho_s_y	(float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the y direction. Defaults to -1, e.g. computed in **init**() and ReInit() assuming one range of hoops.

Parameters

Ec	(float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in **init**() and ReInit().

Exceptions

NegativeValue	b needs to be positive.
NegativeValue	d needs to be positive.
NegativeValue	L needs to be positive.
NegativeValue	e needs to be positive.
PositiveValue	fc needs to be negative.
NegativeValue	D_bars needs to be positive.
NegativeValue	fy needs to be positive.
NegativeValue	Ey needs to be positive.
NegativeValue	D_hoops needs to be positive.
NegativeValue	s needs to be positive.
NegativeValue	fs needs to be positive.
NegativeValue	Es needs to be positive.
NegativeValue	rho_s_x needs to be positive if different from -1.
NegativeValue	rho_s_y needs to be positive if different from -1.
NegativeValue	Ec needs to be positive if different from -1.
WrongDimension	Number of lists in the list bars_position_x needs to be the same of the length of bars_ranges_position_y - 1.
InconsistentGeometry	The sum of the distances for each list in bars_position_x should be equal to the section's width (tol = 5 mm).
InconsistentGeometry	The sum of the distances in bars_ranges_position_y should be equal to the section's depth (tol = 5 mm).
InconsistentGeometry	e should be smaller than half the depth and the width of the section.

Reimplemented in RCSquareShape.

Definition at line 271 of file Section.py.

```
        D_hoops, s, fs, Es, name_tag = "Not Defined", rho_s_x = -1, rho_s_y = -1, Ec = -1):
        """
        The conctructor of the class.

        @param b (float): Width of the section.
        @param d (float): Depth of the section.
        @param L (float): Effective length of the element associated with this section.
            If the panel zone is present, exclude its dimension.
        @param e (float): Concrete cover.
        @param fc (float): Unconfined concrete compressive strength (cylinder test).
        @param D_bars (float): Diameter of the reinforcing bars.
        @param bars_position_x (np.ndarray): Array with a range of aligned vertical reinforcing bars for each row in x direction.
            Distances from border to bar centerline, bar to bar centerlines and
            finally bar centerline to border in the x direction (aligned).
            Starting from the left to right, from the top range to the bottom one.
            The number of bars for each range can vary; in this case, add this argument when defining the array " dtype = object".
        @param bars_ranges_position_y (np.ndarray): Array of dimension 1 with the position or spacing in y of the ranges in bars_position_x.
            Distances from border to range centerlines, range to range centerlines and
            finally range centerline to border in the y direction.
            Starting from the top range to the bottom one.
        @param fy (float): Yield stress for reinforcing bars.
        @param Ey (float): Young modulus for reinforcing bars.
        @param D_hoops (float): Diameter of the hoops.
        @param s (float): Centerline distance for the hoops.
        @param fs (float): Yield stress for the hoops.
        @param Es (float): Young modulus for the hoops
```

```
        @param name_tag (str, optional): A nametag for the section. Defaults to "Not Defined".
        @param rho_s_x (float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the x direction.
            Defaults to -1, e.g. computed in __init__()  and ReInit() assuming one range of hoops.
        @param rho_s_y (float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the y direction.
            Defaults to -1, e.g. computed in __init__()  and ReInit() assuming one range of hoops.
        @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in __init__() and ReInit().

        @exception NegativeValue: b needs to be positive.
        @exception NegativeValue: d needs to be positive.
        @exception NegativeValue: L needs to be positive.
        @exception NegativeValue: e needs to be positive.
        @exception PositiveValue: fc needs to be negative.
        @exception NegativeValue: D_bars needs to be positive.
        @exception NegativeValue: fy needs to be positive.
        @exception NegativeValue: Ey needs to be positive.
        @exception NegativeValue: D_hoops needs to be positive.
        @exception NegativeValue: s needs to be positive.
        @exception NegativeValue: fs needs to be positive.
        @exception NegativeValue: Es needs to be positive.
        @exception NegativeValue: rho_s_x needs to be positive if different from -1.
        @exception NegativeValue: rho_s_y needs to be positive if different from -1.
        @exception NegativeValue: Ec needs to be positive if different from -1.
        @exception WrongDimension: Number of lists in the list bars_position_x needs to be the same of the length of bars_ranges_position_y - 1.
        @exception InconsistentGeometry: The sum of the distances for each list in bars_position_x should be equal to the section's width (tol = 5 mm).
        @exception InconsistentGeometry: The sum of the distances in bars_ranges_position_y should be equal to the section's depth (tol = 5 mm).
        @exception InconsistentGeometry: e should be smaller than half the depth and the width of the section.
        """
        # Check
        if b < 0: raise NegativeValue()
        if d < 0: raise NegativeValue()
        if L < 0: raise NegativeValue()
        if e < 0: raise NegativeValue()
        if fc > 0: raise PositiveValue()
        if D_bars < 0: raise NegativeValue()
        if fy < 0: raise NegativeValue()
        if Ey < 0: raise NegativeValue()
        if D_hoops < 0: raise NegativeValue()
        if s < 0: raise NegativeValue()
        if fs < 0: raise NegativeValue()
        if Es < 0: raise NegativeValue()
        if rho_s_x != -1 and rho_s_x < 0: raise NegativeValue()
        if rho_s_y != -1 and rho_s_y < 0: raise NegativeValue()
        if Ec != -1 and Ec < 0: raise NegativeValue()
        if np.size(bars_position_x) != np.size(bars_ranges_position_y)-1: raise WrongDimension()
        geometry_tol = 5*mm_unit
        for bars in bars_position_x:
            if abs(np.sum(bars) - b) > geometry_tol: raise InconsistentGeometry()
        if abs(np.sum(bars_ranges_position_y)-d) > geometry_tol: raise InconsistentGeometry()
        if e > b/2 or e > d/2: raise InconsistentGeometry()
        warning_min_bars = "!!!!!!! WARNING !!!!!!! The hypothesis of one bar per corner (aligned) is not fullfilled."
        if len(bars_position_x) < 2:
            print(warning_min_bars)
        elif len(bars_position_x[0]) < 3 or len(bars_position_x[-1]) < 3:
            print(warning_min_bars)

        # Arguments
        self.b = b
        self.d = d
        self.L = L
        self.e = e
        self.fc = fc
        self.D_bars = D_bars
        self.bars_position_x = deepcopy(bars_position_x)
        self.bars_ranges_position_y = copy(bars_ranges_position_y)
        self.fy = fy
        self.Ey = Ey
        self.D_hoops = D_hoops
        self.s = s
        self.fs = fs
        self.Es = Es
        self.name_tag = name_tag

        # Initialized the parameters that are dependent from others
        self.ReInit(rho_s_x, rho_s_y, Ec)


```

7.45.3 Member Function Documentation

7.45.3.1 ComputeA()

```
def ComputeA (
                self )
```

Compute the area for a rectangular section.

Returns

float: Total area.

Definition at line 495 of file Section.py.

```
00495     def ComputeA(self):
00496         """
00497         Compute the area for a rectangular section.
00498 
00499         @returns float: Total area.
00500         """
00501         return self.b * self.d
00502 
00503 
```

7.45.3.2 ComputeAc()

```
def ComputeAc (
                self )
```

Compute the confined area (area inside the centerline of the hoops, according to Mander et Al.

1988).

Returns

float: Confined area.

Definition at line 504 of file Section.py.

```
00504     def ComputeAc(self):
00505         """
00506         Compute the confined area (area inside the centerline of the hoops, according to Mander et Al.
       1988).
00507 
00508         @returns float: Confined area.
00509         """
00510         return self.bc * self.dc
00511 
00512 
```

7.45.3.3 ComputeEc()

```
def ComputeEc (
        self )
```

Compute Ec using the formula from Mander et Al.

1988.

Returns

float: Young modulus of concrete.

Definition at line 485 of file Section.py.

```
00485     def ComputeEc(self):
00486         """
00487         Compute Ec using the formula from Mander et Al. 1988.
00488 
00489         @returns float: Young modulus of concrete.
00490         """
00491 
00492         return 5000.0 * math.sqrt(-self.fc/MPa_unit) * MPa_unit
00493 
00494 
```

7.45.3.4 ComputeIy()

```
def ComputeIy (
        self )
```

Compute the moment of inertia of the rectangular section with respect to the strong axis.

Returns

float: Moment of inertia (strong axis)

Definition at line 513 of file Section.py.

```
00513     def ComputeIy(self):
00514         """
00515         Compute the moment of inertia of the rectangular section with respect to the strong axis.
00516 
00517         @returns float: Moment of inertia (strong axis)
00518         """
00519         return self.b * self.d**3 / 12.0
00520 
00521 
```

7.45.3.5 ComputeIz()

```
def ComputeIz (
        self )
```

Compute the moment of inertia of the rectangular section with respect to the weak axis.

Returns

float: Moment of inertia (weak axis)

Definition at line 522 of file Section.py.

```
00522     def ComputeIz(self):
00523         """
00524         Compute the moment of inertia of the rectangular section with respect to the weak axis.
00525 
00526         @returns float: Moment of inertia (weak axis)
00527         """
00528         return self.d * self.b**3 / 12.0
00529 
00530 
```

7.45.3.6 ComputeNrBars()

```
def ComputeNrBars (
              self )
```

Compute the number of vertical bars in the array bars_position_x (note that this list of lists can have different list sizes).

Returns

int: Number of vertical reinforcing bars.

Definition at line 472 of file Section.py.

```
00472     def ComputeNrBars(self):
00473         """
00474         Compute the number of vertical bars in the array bars_position_x (note that this list of lists
      can have different list sizes).
00475 
00476         @returns int: Number of vertical reinforcing bars.
00477         """
00478         nr_bars = 0
00479         for range in self.bars_position_x:
00480             nr_bars += np.size(range)-1
00481 
00482         return nr_bars
00483 
00484 
```

7.45.3.7 ReInit()

```
def ReInit (
              self,
              rho_s_x = -1,
              rho_s_y = -1,
              Ec = -1 )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

rho_s_x	(float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the x direction. Defaults to -1, e.g. computed assuming one range of hoops.
rho_s_y	(float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the y direction. Defaults to -1, e.g. computed assuming one range of hoops.
Ec	(float, optional): Young modulus for concrete. Defaults to -1, e.g. computed according to Mander et Al. 1988.

Definition at line 374 of file Section.py.

```
00374     def ReInit(self, rho_s_x = -1, rho_s_y = -1, Ec = -1):
00375         """
00376         Implementation of the homonym abstract method.
00377         See parent class DataManagement for detailed information.
00378 
00379         @param rho_s_x (float, optional): Ratio of the transversal area of the hoops to the associated
      concrete area in the x direction.
00380             Defaults to -1, e.g. computed assuming one range of hoops.
```

```
        @param rho_s_y (float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the y direction.
            Defaults to -1, e.g. computed assuming one range of hoops.
        @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed according to Mander et Al. 1988.
        """
        # Precompute some members
        self.cl_hoops = self.e + self.D_hoops/2.0 # centerline distance from the border of the extreme confining hoops
        self.cl_bars = self.e + self.D_bars/2.0 + self.D_hoops # centerline distance from the border of the corner bars
        self.bc = self.b - self.cl_hoops*2
        self.dc = self.d - self.cl_hoops*2
        self.As = ComputeACircle(self.D_hoops)

        # Arguments
        self.rho_s_x = 2.0*ComputeRho(self.As, 1, self.bc*self.s) if rho_s_x == -1 else rho_s_x
        self.rho_s_y = 2.0*ComputeRho(self.As, 1, self.dc*self.s) if rho_s_y == -1 else rho_s_y
        self.Ec = self.ComputeEc() if Ec == -1 else Ec

        # Members
        self.nr_bars = self.ComputeNrBars()
        self.A = self.ComputeA()
        self.Ac = self.ComputeAc()
        self.Ay = ComputeACircle(self.D_bars)
        self.rho_bars = ComputeRho(self.Ay, self.nr_bars, self.A)
        self.Iy = self.ComputeIy()
        self.Iz = self.ComputeIz()

        # Data storage for loading/saving
        self.UpdateStoredData()
```

7.45.3.8 ShowInfo()

```
def ShowInfo (
                self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 448 of file Section.py.

```
00448     def ShowInfo(self):
00449         """
00450         Implementation of the homonym abstract method.
00451         See parent class DataManagement for detailed information.
00452         """
00453         print("")
00454         print("Requested info for RC rectangular section of name tag = {}".format(self.name_tag))
00455         print("Width of the section b = {} mm".format(self.b/mm_unit))
00456         print("Depth of the section d = {} mm".format(self.d/mm_unit))
00457         print("Concrete cover e = {} mm".format(self.e/mm_unit))
00458         print("Concrete area A = {} mm2".format(self.A/mm2_unit))
00459         print("Core concrete area Ac = {} mm2".format(self.Ac/mm2_unit))
00460         print("Unconfined concrete compressive strength fc = {} MPa".format(self.fc/MPa_unit))
00461         print("Young modulus for concrete Ec = {} GPa".format(self.Ec/GPa_unit))
00462         print("Diameter of the reinforcing bars D_bars = {} mm and area of one bar Ay = {} mm2 with {} bars".format(self.D_bars/mm_unit, self.Ay/mm2_unit, self.nr_bars))
00463         print("Diameter of the hoops D_hoops = {} mm and area of one stirrup As = {} mm2".format(self.D_hoops/mm_unit, self.As/mm2_unit))
00464         print("Ratio of area of longitudinal reinforcement to area of concrete section rho_bars = {}".format(self.rho_bars))
00465         print("Ratio of area of lateral reinforcement to lateral area of concrete section in x rho_s_x = {} ".format(self.rho_s_x))
00466         print("Ratio of area of lateral reinforcement to lateral area of concrete section in y rho_s_y = {} ".format(self.rho_s_y))
00467         print("Moment of inertia of the circular section (strong axis) Iy = {} mm4".format(self.Iy/mm4_unit))
00468         print("Moment of inertia of the circular section (weak axis) Iz = {} mm4".format(self.Iz/mm4_unit))
00469         print("")
00470
00471
```

7.45.3.9 UpdateStoredData()

```
def UpdateStoredData (
              self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 410 of file Section.py.

```
00410     def UpdateStoredData(self):
00411         """
00412         Implementation of the homonym abstract method.
00413         See parent class DataManagement for detailed information.
00414         """
00415         self.data = [["INFO_TYPE", "RCRectShape"], # Tag for differentiating different data
00416             ["name_tag", self.name_tag],
00417             ["b", self.b],
00418             ["d", self.d],
00419             ["bc", self.bc],
00420             ["dc", self.dc],
00421             ["L", self.L],
00422             ["e", self.e],
00423             ["A", self.A],
00424             ["Ac", self.Ac],
00425             ["Iy", self.Iy],
00426             ["Iz", self.Iz],
00427             ["fc", self.fc],
00428             ["Ec", self.Ec],
00429             ["D_bars", self.D_bars],
00430             ["nr_bars", self.nr_bars],
00431             ["Ay", self.Ay],
00432             ["bars_position_x", self.bars_position_x],
00433             ["bars_ranges_position_y", self.bars_ranges_position_y],
00434             ["rho_bars", self.rho_bars],
00435             ["cl_bars", self.cl_bars],
00436             ["fy", self.fy],
00437             ["Ey", self.Ey],
00438             ["D_hoops", self.D_hoops],
00439             ["s", self.s],
00440             ["As", self.As],
00441             ["rho_s_x", self.rho_s_x],
00442             ["rho_s_y", self.rho_s_y],
00443             ["cl_hoops", self.cl_hoops],
00444             ["fs", self.fs],
00445             ["Es", self.Es]]
00446
00447
```

7.45.4 Member Data Documentation

7.45.4.1 A

```
A
```

Definition at line 399 of file Section.py.

7.45.4.2 Ac

```
Ac
```

Definition at line 400 of file Section.py.

7.45.4.3 As

```
As
```

Definition at line 390 of file Section.py.

7.45.4.4 Ay

```
Ay
```

Definition at line 401 of file Section.py.

7.45.4.5 b

```
b
```

Definition at line 354 of file Section.py.

7.45.4.6 bars_position_x

```
bars_position_x
```

Definition at line 360 of file Section.py.

7.45.4.7 bars_ranges_position_y

```
bars_ranges_position_y
```

Definition at line 361 of file Section.py.

7.45.4.8 bc

```
bc
```

Definition at line 388 of file Section.py.

7.45.4.9 cl_bars

`cl_bars`

Definition at line 387 of file Section.py.

7.45.4.10 cl_hoops

`cl_hoops`

Definition at line 386 of file Section.py.

7.45.4.11 d

`d`

Definition at line 355 of file Section.py.

7.45.4.12 D_bars

`D_bars`

Definition at line 359 of file Section.py.

7.45.4.13 D_hoops

`D_hoops`

Definition at line 364 of file Section.py.

7.45.4.14 data

`data`

Definition at line 415 of file Section.py.

7.45.4.15 dc

`dc`

Definition at line 389 of file Section.py.

7.45.4.16 e

`e`

Definition at line 357 of file Section.py.

7.45.4.17 Ec

`Ec`

Definition at line 395 of file Section.py.

7.45.4.18 Es

`Es`

Definition at line 367 of file Section.py.

7.45.4.19 Ey

`Ey`

Definition at line 363 of file Section.py.

7.45.4.20 fc

`fc`

Definition at line 358 of file Section.py.

7.45.4.21 fs

```
fs
```

Definition at line 366 of file Section.py.

7.45.4.22 fy

```
fy
```

Definition at line 362 of file Section.py.

7.45.4.23 Iy

```
Iy
```

Definition at line 403 of file Section.py.

7.45.4.24 Iz

```
Iz
```

Definition at line 404 of file Section.py.

7.45.4.25 L

```
L
```

Definition at line 356 of file Section.py.

7.45.4.26 name_tag

```
name_tag
```

Definition at line 368 of file Section.py.

7.45.4.27 nr_bars

`nr_bars`

Definition at line 398 of file Section.py.

7.45.4.28 rho_bars

`rho_bars`

Definition at line 402 of file Section.py.

7.45.4.29 rho_s_x

`rho_s_x`

Definition at line 393 of file Section.py.

7.45.4.30 rho_s_y

`rho_s_y`

Definition at line 394 of file Section.py.

7.45.4.31 s

`s`

Definition at line 365 of file Section.py.

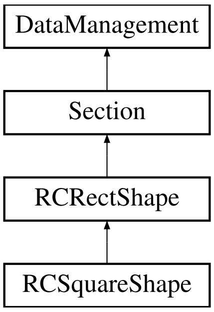
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Section.py

7.46 RCSquareShape Class Reference

Class that is the children of RCRectShape and cover the specific case of square RC sections.

Inheritance diagram for RCSquareShape:



Public Member Functions

- def __init__ (self, b, L, e, fc, D_bars, np.ndarray bars_position_x, np.ndarray bars_ranges_position_y, fy, Ey, D_hoops, s, fs, Es, name_tag="Not Defined", rho_s_x=-1, rho_s_y=-1, Ec=-1)

    Constructor of the class.

Additional Inherited Members

7.46.1 Detailed Description

Class that is the children of RCRectShape and cover the specific case of square RC sections.

Parameters

RCRectShape	Parent class.

Definition at line 531 of file Section.py.

7.46.2 Constructor & Destructor Documentation

7.46.2.1 __init__()

```
def __init__ (
              self,
              b,
```

```
    L,
    e,
    fc,
    D_bars,
   np.ndarray bars_position_x,
   np.ndarray bars_ranges_position_y,
    fy,
    Ey,
    D_hoops,
    s,
    fs,
    Es,
    name_tag = "Not Defined",
    rho_s_x = -1,
    rho_s_y = -1,
    Ec = -1 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the specific case of a aquare RC section.

Parameters

b	(float): Width/depth of the section.
L	(float): Effective length of the element associated with this section. If the panel zone is present, exclude its dimension.
e	(float): Concrete cover.
fc	(float): Unconfined concrete compressive strength (cylinder test).
D_bars	(float): Diameter of the reinforcing bars.
bars_position_x	(np.ndarray): Distances from border to bar centerline, bar to bar centerlines and finally bar centerline to border in the x direction (aligned). Starting from the left to right, from the top range to the bottom one. The number of bars for each range can vary; in this case, add this argument when defining the array " dtype = object".
bars_ranges_position↩_y	(np.ndarray): Distances from border to range centerlines, range to range centerlines and finally range centerline to border in the y direction. Starting from the top range to the bottom one.
fy	(float): Yield stress for reinforcing bars.
Ey	(float): Young modulus for reinforcing bars.
D_hoops	(float): Diameter of the hoops.
s	(float): Vertical centerline spacing between hoops.
fs	(float): Yield stress for the hoops.
Es	(float): Young modulus for the hoops
name_tag	(str, optional): A nametag for the section. Defaults to "Not Defined".
rho_s_x	(float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the x direction. Defaults to -1, e.g. computed in **init**() and ReInit() assuming one range of hoops.
rho_s_y	(float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the y direction. Defaults to -1, e.g. computed in **init**() and ReInit() assuming one range of hoops.
Ec	(float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in **init**() and ReInit().

Reimplemented from RCRectShape.

Definition at line 537 of file Section.py.

```
00537     def __init__(self, b, L, e, fc, D_bars, bars_position_x: np.ndarray, bars_ranges_position_y: np.ndarray, fy, Ey, D_hoops, s, fs, Es, name_tag="Not Defined", rho_s_x=-1, rho_s_y=-1, Ec=-1):
00538         """
00539         Constructor of the class. It passes the arguments into the parent class to generate the specific case of a aquare RC section.
00540 
00541         @param b (float): Width/depth of the section.
00542         @param L (float): Effective length of the element associated with this section.
00543             If the panel zone is present, exclude its dimension.
00544         @param e (float): Concrete cover.
00545         @param fc (float): Unconfined concrete compressive strength (cylinder test).
00546         @param D_bars (float): Diameter of the reinforcing bars.
00547         @param bars_position_x (np.ndarray):  Distances from border to bar centerline, bar to bar centerlines and
00548             finally bar centerline to border in the x direction (aligned).
00549             Starting from the left to right, from the top range to the bottom one.
00550             The number of bars for each range can vary; in this case, add this argument when defining the array " dtype = object".
00551         @param bars_ranges_position_y (np.ndarray): Distances from border to range centerlines, range to range centerlines and
00552             finally range centerline to border in the y direction.
00553             Starting from the top range to the bottom one.
00554         @param fy (float): Yield stress for reinforcing bars.
00555         @param Ey (float): Young modulus for reinforcing bars.
00556         @param D_hoops (float): Diameter of the hoops.
00557         @param s (float): Vertical centerline spacing between hoops.
00558         @param fs (float): Yield stress for the hoops.
00559         @param Es (float): Young modulus for the hoops
00560         @param name_tag (str, optional): A nametag for the section. Defaults to "Not Defined".
00561         @param rho_s_x (float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the x direction.
00562             Defaults to -1, e.g. computed in __init__()  and ReInit() assuming one range of hoops.
00563         @param rho_s_y (float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the y direction.
00564             Defaults to -1, e.g. computed in __init__()  and ReInit() assuming one range of hoops.
00565         @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in __init__() and ReInit().
00566         """
00567         super().__init__(b, b, L, e, fc, D_bars, bars_position_x, bars_ranges_position_y, fy, Ey, D_hoops, s, fs, Es, name_tag, rho_s_x, rho_s_y, Ec)
00568 
00569 
```

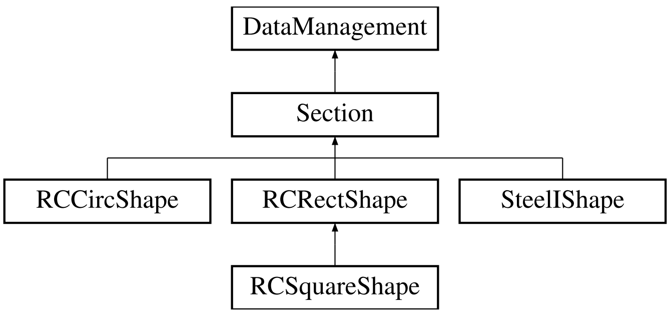
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Section.py

7.47 Section Class Reference

Parent abstract class for the storage and manipulation of a section's information (mechanical and geometrical parameters, etc).

Inheritance diagram for Section:



7.47.1 Detailed Description

Parent abstract class for the storage and manipulation of a section's information (mechanical and geometrical parameters, etc).

Parameters

DataManagement	Parent abstract class.

Definition at line 13 of file Section.py.

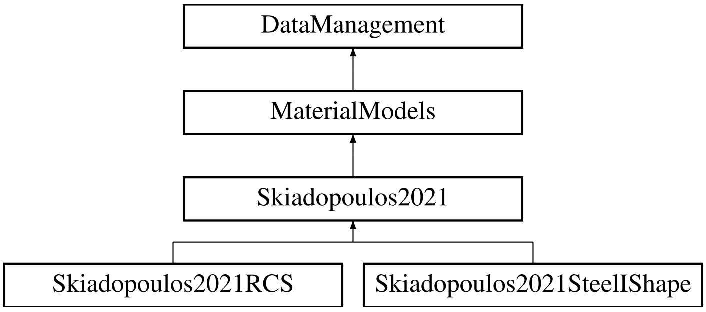
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Section.py

7.48 Skiadopoulos2021 Class Reference

Class that stores funcions and material properties of a steel double symmetric I-shape profile with Skiadopoulos 2021 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.

Inheritance diagram for Skiadopoulos2021:



Public Member Functions

- def __init__ (self, int ID, d_c, bf_c, tf_c, I_c, d_b, tf_b, Fy, E, t_p, t_dp=0.0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0.0, dmg2=0.0, beta=0.0, safety_factor=False, t_fbp=0)

  Constructor of the class.
- def CheckApplicability (self)

  Implementation of the homonym abstract method.
- def Hysteretic (self)

  Generate the material model Hysteretic (Skiadopoulos 2021) using the computed parameters.
- def ReInit (self)

  Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False)

  Implementation of the homonym abstract method.
- def UpdateStoredData (self)

  Implementation of the homonym abstract method.

Public Attributes

- a_s
- beam_section_name_tag
- beta
- bf_c
- Cf1
- Cf4
- Cf6
- col_section_name_tag
- Cw1
- Cw4
- Cw6
- d_b
- d_c
- data
- dmg1
- dmg2
- E
- Fy
- G
- Gamma_1
- Gamma_4
- Gamma_6
- I_c
- ID
- Initialized
- Kb
- Kbf
- Ke
- Kf
- Kf_Ke
- Ks
- Ksf
- M1
- M4
- M6
- pinchx
- pinchy
- Ry
- t_dp
- t_fbp
- t_p
- t_pz
- tf_b
- tf_c
- V1
- V4
- V6

Static Public Attributes

- list Cf1_tests = [0.035, 0.035, 0.033, 0.031, 0.018, 0.015, 0.013, 0.009, 0.009, 0.010, 0.010]
- list Cf4_tests = [0.145, 0.145, 0.123, 0.111, 0.069, 0.040, 0.040, 0.018, 0.010, 0.012, 0.012]
- list Cf6_tests = [0.165, 0.1650, 0.1400, 0.1275, 0.0800, 0.0500, 0.0500, 0.0180, 0.0140, 0.0120, 0.0120]
- list Cw1_tests = [0.96, 0.96, 0.955, 0.94, 0.93, 0.90, 0.89, 0.89, 0.88, 0.88, 0.88]
- list Cw4_tests = [1.145, 1.145, 1.140, 1.133, 1.120, 1.115, 1.115, 1.11, 1.10, 1.10, 1.10]
- list Cw6_tests = [1.205, 1.2050, 1.2000, 1.1925, 1.1740, 1.1730, 1.1720, 1.1690, 1.1670, 1.1650, 1.1650]
- list Kf_Ke_tests = [1.000, 0.153, 0.120, 0.090, 0.059, 0.031, 0.019, 0.009, 0.005, 0.004, 0.000]

7.48.1 Detailed Description

Class that stores funcions and material properties of a steel double symmetric I-shape profile with Skiadopoulos 2021 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.

The material model is valid only if the column is continuous. For more information about the empirical model for the computation of the parameters, see Skiadopoulos et Al. 2021. The vectors that forms the matrix used to compute the material model parameters (Kf_Ke_tests, Cw1_tests, Cf1_tests, Cw4_tests, Cf4_tests, Cw6_tests, Cf6_tests) are used as global throughout the class to optimise the program (given the fact that is constant everytime).

Parameters

MaterialModels	Parent abstract class.

Definition at line 780 of file MaterialModels.py.

7.48.2 Constructor & Destructor Documentation

7.48.2.1 __init__()

```
def __init__ (
        self,
        int ID,
        d_c,
        bf_c,
        tf_c,
        I_c,
        d_b,
        tf_b,
        Fy,
        E,
        t_p,
        t_dp = 0.0,
        a_s = 0.03,
        pinchx = 0.25,
        pinchy = 0.75,
        dmg1 = 0.0,
        dmg2 = 0.0,
        beta = 0.0,
```

safety_factor = False,
t_fbp = 0)

Constructor of the class.

Parameters

ID	(int): Unique material model ID.
d_c	(float): Column depth.
bf_c	(float): Column flange width.
tf_c	(float): Column flange thickness.
I_c	(float): Column moment of inertia (strong axis).
d_b	(float): Beam depth.
tf_b	(float): Beam flange thickness.
Fy	(float): Yield strength (if assume continous column, Fy of the web).
E	(float): Young modulus.
t_p	(float): Panel zone thickness.
t_dp	(float, optional): Doubler plate thickness. Defaults to 0.0.
a_s	(float, optional): Strain hardening. Defaults to 0.03.
pinchx	(float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25
pinchy	(float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75
dmg1	(float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
dmg2	(float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
beta	(float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
safety_factor	(bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.
t_fbp	(float, optional): Thickness of the face bearing plate (if present). Defaults to 0.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	d_c needs to be positive.
NegativeValue	bf_c needs to be positive.
NegativeValue	tf_c needs to be positive.
NegativeValue	d_b needs to be positive.
NegativeValue	tf_b needs to be positive.
NegativeValue	Fy needs to be positive.
NegativeValue	E needs to be positive.
NegativeValue	t_p needs to be positive.
NegativeValue	a_s needs to be positive.

Reimplemented in Skiadopoulos2021RCS, and Skiadopoulos2021SteelIShape.

Definition at line 808 of file MaterialModels.py.

```
        t_dp = 0.0, a_s = 0.03, pinchx = 0.25, pinchy = 0.75, dmg1 = 0.0, dmg2 = 0.0, beta = 0.0,
    safety_factor = False, t_fbp = 0):
        """
        Constructor of the class.

        @param ID (int): Unique material model ID.
        @param d_c (float): Column depth.
        @param bf_c (float): Column flange width.
        @param tf_c (float): Column flange thickness.
        @param I_c (float): Column moment of inertia (strong axis).
        @param d_b (float): Beam depth.
        @param tf_b (float): Beam flange thickness.
        @param Fy (float): Yield strength (if assume continous column, Fy of the web).
        @param E (float): Young modulus.
        @param t_p (float): Panel zone thickness.
```

```
        @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
        @param a_s (float, optional): Strain hardening. Defaults to 0.03.
        @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25
        @param pinchy (float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75
        @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
        @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
        @param beta (float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
        @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.
        @param t_fbp (float, optional): Thickness of the face bearing plate (if present). Defaults to 0.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: d_c needs to be positive.
        @exception NegativeValue: bf_c needs to be positive.
        @exception NegativeValue: tf_c needs to be positive.
        @exception NegativeValue: d_b needs to be positive.
        @exception NegativeValue: tf_b needs to be positive.
        @exception NegativeValue: Fy needs to be positive.
        @exception NegativeValue: E needs to be positive.
        @exception NegativeValue: t_p needs to be positive.
        @exception NegativeValue: a_s needs to be positive.
        """
        # Check
        if ID < 1: raise NegativeValue()
        if d_c < 0: raise NegativeValue()
        if bf_c < 0: raise NegativeValue()
        if tf_c < 0: raise NegativeValue()
        if d_b < 0: raise NegativeValue()
        if tf_b < 0: raise NegativeValue()
        if Fy < 0: raise NegativeValue()
        if E < 0: raise NegativeValue()
        if t_p < 0: raise NegativeValue()
        if a_s < 0: raise NegativeValue()
        if t_fbp < 0: raise NegativeValue()

        # Arguments
        self.ID = ID
        self.d_c = d_c
        self.bf_c = bf_c
        self.tf_c = tf_c
        self.I_c = I_c
        self.d_b = d_b
        self.tf_b = tf_b
        self.Fy = Fy
        self.E = E
        self.t_p = t_p
        self.t_dp = t_dp
        self.a_s = a_s
        self.pinchx = pinchx
        self.pinchy = pinchy
        self.dmg1 = dmg1
        self.dmg2 = dmg2
        self.beta = beta
        if safety_factor:
            self.Ry = 1.2
        else:
            self.Ry = 1.0
        self.t_fbp = t_fbp

        # Initialized the parameters that are dependent from others
        self.beam_section_name_tag = "None"
        self.col_section_name_tag = "None"
        self.Initialized = False
        self.ReInit()
```

7.48.3 Member Function Documentation

7.48.3.1 CheckApplicability()

```
def CheckApplicability (
        self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Reimplemented from MaterialModels.

Definition at line 1025 of file MaterialModels.py.

```
    def CheckApplicability(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        Check = True
        # No checks
        if not Check:
            print("The validity of the equations is not fullfilled.")
            print("!!!!!!! WARNING !!!!!!! Check material model of Skiadopoulos 2021, ID=", self.ID)
            print("")


```

7.48.3.2 Hysteretic()

```
def Hysteretic (
        self )
```

Generate the material model Hysteretic (Skiadopoulos 2021) using the computed parameters.

See _Hysteretic function for more information.

Definition at line 1038 of file MaterialModels.py.

```
    def Hysteretic(self):
        """
        Generate the material model Hysteretic (Skiadopoulos 2021) using the computed parameters.
        See _Hysteretic function for more information.
        """
        _Hysteretic(self.ID, self.M1, self.Gamma_1, self.M4, self.Gamma_4, self.M6, self.Gamma_6,
            self.pinchx, self.pinchy, self.dmg1, self.dmg2, self.beta)
        self.Initialized = True
        self.UpdateStoredData()


```

7.48.3.3 ReInit()

```
def ReInit (
        self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 889 of file MaterialModels.py.

```
    def ReInit(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        # Check applicability
        self.CheckApplicability()

        # Memebers
```

```
        if self.beam_section_name_tag != "None": self.beam_section_name_tag = self.beam_section_name_tag + " (modified)"
        if self.col_section_name_tag != "None": self.col_section_name_tag = self.col_section_name_tag + " (modified)"
        self.t_pz = self.t_p + self.t_dp
        self.G = self.E/(2.0 * (1.0 + 0.30)) # Shear Modulus

        # Refined computation of the parameters for the backbone curve for the panel zone spring (Skiadopoulos et al. (2021))
        # Panel Zone Elastic Stiffness
        self.Ks = self.t_pz*(self.d_c-self.tf_c)*self.G
        self.Kb = 12.0*self.E*(self.I_c+self.t_dp*(self.d_c-2.0*self.tf_c)**3/12.0)/(self.d_b-0)**2
        self.Ke = self.Ks*self.Kb/(self.Ks+self.Kb)

        # Column Flange Stiffness
        self.Ksf = 2.0*((self.tf_c+self.t_fbp)*self.bf_c*self.G)
        self.Kbf = 2.0*(12.0*self.E*self.bf_c*(self.tf_c**3+self.t_fbp**3)/12.0/(self.d_b-0)**2)
        self.Kf = self.Ksf*self.Kbf/(self.Ksf+self.Kbf)

        # Kf/Ke Calculation for Panel Zone Categorization
        self.Kf_Ke = self.Kf/self.Ke

        # Panel Zone Strength Coefficients (results from tests for a_w_eff and a_f_eff)
        self.Cw1 = np.interp(self.Kf_Ke, Kf_Ke_tests, Cw1_tests)
        self.Cf1 = np.interp(self.Kf_Ke, Kf_Ke_tests, Cf1_tests)
        self.Cw4 = np.interp(self.Kf_Ke, Kf_Ke_tests, Cw4_tests)
        self.Cf4 = np.interp(self.Kf_Ke, Kf_Ke_tests, Cf4_tests)
        self.Cw6 = np.interp(self.Kf_Ke, Kf_Ke_tests, Cw6_tests)
        self.Cf6 = np.interp(self.Kf_Ke, Kf_Ke_tests, Cf6_tests)

        # Panel Zone Model
        self.V1 = self.Fy*self.Ry/math.sqrt(3)*(self.Cw1*(self.d_c-self.tf_c)*self.t_pz + self.Cf1*2*(self.bf_c-self.t_p)*self.tf_c)
        self.V4 = self.Fy*self.Ry/math.sqrt(3)*(self.Cw4*(self.d_c-self.tf_c)*self.t_pz + self.Cf4*2*(self.bf_c-self.t_p)*self.tf_c)
        self.V6 = self.Fy*self.Ry/math.sqrt(3)*(self.Cw6*(self.d_c-self.tf_c)*self.t_pz + self.Cf6*2*(self.bf_c-self.t_p)*self.tf_c)

        self.M1 = self.V1*(self.d_b-self.tf_b)
        self.M4 = self.V4*(self.d_b-self.tf_b)
        self.M6 = self.V6*(self.d_b-self.tf_b)

        self.Gamma_1 = self.V1/self.Ke
        self.Gamma_4 = 4*self.Gamma_1
        self.Gamma_6 = 6*self.Gamma_1

        # Data storage for loading/saving
        self.UpdateStoredData()
```

7.48.3.4 ShowInfo()

```
def ShowInfo (
        self,
        plot = False,
        block = False )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the material model. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 989 of file MaterialModels.py.

```
    def ShowInfo(self, plot = False, block = False):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        """
        print("")
        print("Requested info for Skiadopoulos 2021 material model Parameters, ID = {}".format(self.ID))
        print("Sections associated, column: {} ".format(self.col_section_name_tag))
        print("Sections associated, beam: {} ".format(self.beam_section_name_tag))
        print("Gamma_1 = {} rad".format(self.Gamma_1))
        print("Gamma_4 = {} rad".format(self.Gamma_4))
        print("Gamma_6 = {} rad".format(self.Gamma_6))
        print("M1 = {} kNm".format(self.M1/kNm_unit))
        print("M4 = {} kNm".format(self.M4/kNm_unit))
        print("M6 = {} kNm".format(self.M6/kNm_unit))
        print("")

        if plot:
            # Data for plotting
            x_axis = np.array([0.0, self.Gamma_1, self.Gamma_4, self.Gamma_6])
            y_axis = np.array([0.0, self.M1, self.M4, self.M6])/kNm_unit

            fig, ax = plt.subplots()
            ax.plot(x_axis, y_axis, 'k-')

            ax.set(xlabel='Rotation [rad]', ylabel='Moment [kNm]',
                title='Skiadopoulos 2021 material model (ID={})'.format(self.ID))
            ax.grid()

            if block:
                plt.show()


```

7.48.3.5 UpdateStoredData()

```
def UpdateStoredData (
            self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 942 of file MaterialModels.py.

```
00942     def UpdateStoredData(self):
00943         """
00944         Implementation of the homonym abstract method.
00945         See parent class DataManagement for detailed information.
00946         """
00947         self.data = [["INFO_TYPE", "Skiadopoulos2021"], # Tag for differentiating different data
00948             ["ID", self.ID],
00949             ["beam_section_name_tag", self.beam_section_name_tag],
00950             ["col_section_name_tag", self.col_section_name_tag],
00951             ["d_c", self.d_c],
00952             ["bf_c", self.bf_c],
00953             ["tf_c", self.tf_c],
00954             ["I_c", self.I_c],
00955             ["d_b", self.d_b],
00956             ["tf_b", self.tf_b],
00957             ["Fy", self.Fy],
00958             ["E", self.E],
00959             ["G", self.G],
00960             ["t_p", self.t_p],
00961             ["t_dp", self.t_dp],
00962             ["t_pz", self.t_pz],
00963             ["a_s", self.a_s],
00964             ["pinchx", self.pinchx],
00965             ["pinchy", self.pinchy],
00966             ["dmg1", self.dmg1],
00967             ["dmg2", self.dmg2],
```

```
            ["beta", self.beta],
            ["Ry", self.Ry],
            ["Ks", self.Ks],
            ["Kb", self.Kb],
            ["Ke", self.Ke],
            ["Ksf", self.Ksf],
            ["Kbf", self.Kbf],
            ["Kf", self.Kf],
            ["Kf_Ke", self.Kf_Ke],
            ["V1", self.V1],
            ["V4", self.V4],
            ["V6", self.V6],
            ["M1", self.M1],
            ["M4", self.M4],
            ["M6", self.M6],
            ["Gamma_1", self.Gamma_1],
            ["Gamma_4", self.Gamma_4],
            ["Gamma_6", self.Gamma_6],
            ["Initialized", self.Initialized]]


```

7.48.4 Member Data Documentation

7.48.4.1 a_s

`a_s`

Definition at line 869 of file MaterialModels.py.

7.48.4.2 beam_section_name_tag

`beam_section_name_tag`

Definition at line 882 of file MaterialModels.py.

7.48.4.3 beta

`beta`

Definition at line 874 of file MaterialModels.py.

7.48.4.4 bf_c

`bf_c`

Definition at line 860 of file MaterialModels.py.

7.48.4.5 Cf1

```
Cf1
```

Definition at line 919 of file MaterialModels.py.

7.48.4.6 Cf1_tests

```
list Cf1_tests = [0.035, 0.035, 0.033, 0.031, 0.018, 0.015, 0.013, 0.009, 0.009, 0.010, 0.010]
[static]
```

Definition at line 797 of file MaterialModels.py.

7.48.4.7 Cf4

```
Cf4
```

Definition at line 921 of file MaterialModels.py.

7.48.4.8 Cf4_tests

```
list Cf4_tests = [0.145, 0.145, 0.123, 0.111, 0.069, 0.040, 0.040, 0.018, 0.010, 0.012, 0.012]
[static]
```

Definition at line 801 of file MaterialModels.py.

7.48.4.9 Cf6

```
Cf6
```

Definition at line 923 of file MaterialModels.py.

7.48.4.10 Cf6_tests

```
list Cf6_tests = [0.165, 0.1650, 0.1400, 0.1275, 0.0800, 0.0500, 0.0500, 0.0180, 0.0140, 0.↩
0120, 0.0120]  [static]
```

Definition at line 805 of file MaterialModels.py.

7.48.4.11 col_section_name_tag

```
col_section_name_tag
```

Definition at line 883 of file MaterialModels.py.

7.48.4.12 Cw1

```
Cw1
```

Definition at line 918 of file MaterialModels.py.

7.48.4.13 Cw1_tests

```
list Cw1_tests = [0.96, 0.96, 0.955, 0.94, 0.93, 0.90, 0.89, 0.89, 0.88, 0.88, 0.88] [static]
```

Definition at line 795 of file MaterialModels.py.

7.48.4.14 Cw4

```
Cw4
```

Definition at line 920 of file MaterialModels.py.

7.48.4.15 Cw4_tests

```
list Cw4_tests = [1.145, 1.145, 1.140, 1.133, 1.120, 1.115, 1.115, 1.11, 1.10, 1.10, 1.10]
[static]
```

Definition at line 799 of file MaterialModels.py.

7.48.4.16 Cw6

```
Cw6
```

Definition at line 922 of file MaterialModels.py.

7.48.4.17 Cw6_tests

```
list Cw6_tests = [1.205, 1.2050, 1.2000, 1.1925, 1.1740, 1.1730, 1.1720, 1.1690, 1.1670, 1.←
1650, 1.1650] [static]
```

Definition at line 803 of file MaterialModels.py.

7.48.4.18 d_b

```
d_b
```

Definition at line 863 of file MaterialModels.py.

7.48.4.19 d_c

```
d_c
```

Definition at line 859 of file MaterialModels.py.

7.48.4.20 data

```
data
```

Definition at line 947 of file MaterialModels.py.

7.48.4.21 dmg1

```
dmg1
```

Definition at line 872 of file MaterialModels.py.

7.48.4.22 dmg2

```
dmg2
```

Definition at line 873 of file MaterialModels.py.

7.48.4.23 E

```
E
```

Definition at line 866 of file MaterialModels.py.

7.48.4.24 Fy

```
Fy
```

Definition at line 865 of file MaterialModels.py.

7.48.4.25 G

```
G
```

Definition at line 901 of file MaterialModels.py.

7.48.4.26 Gamma_1

```
Gamma_1
```

Definition at line 934 of file MaterialModels.py.

7.48.4.27 Gamma_4

```
Gamma_4
```

Definition at line 935 of file MaterialModels.py.

7.48.4.28 Gamma_6

```
Gamma_6
```

Definition at line 936 of file MaterialModels.py.

7.48.4.29 I_c

```
I_c
```

Definition at line 862 of file MaterialModels.py.

7.48.4.30 ID

```
ID
```

Definition at line 858 of file MaterialModels.py.

7.48.4.31 Initialized

```
Initialized
```

Definition at line 884 of file MaterialModels.py.

7.48.4.32 Kb

```
Kb
```

Definition at line 906 of file MaterialModels.py.

7.48.4.33 Kbf

```
Kbf
```

Definition at line 911 of file MaterialModels.py.

7.48.4.34 Ke

```
Ke
```

Definition at line 907 of file MaterialModels.py.

7.48.4.35 Kf

```
Kf
```

Definition at line 912 of file MaterialModels.py.

7.48.4.36 Kf_Ke

```
Kf_Ke
```

Definition at line 915 of file MaterialModels.py.

7.48.4.37 Kf_Ke_tests

```
list Kf_Ke_tests = [1.000, 0.153, 0.120, 0.090, 0.059, 0.031, 0.019, 0.009, 0.005, 0.004, 0.↩
000]  [static]
```

Definition at line 793 of file MaterialModels.py.

7.48.4.38 Ks

```
Ks
```

Definition at line 905 of file MaterialModels.py.

7.48.4.39 Ksf

```
Ksf
```

Definition at line 910 of file MaterialModels.py.

7.48.4.40 M1

```
M1
```

Definition at line 930 of file MaterialModels.py.

7.48.4.41 M4

```
M4
```

Definition at line 931 of file MaterialModels.py.

7.48.4.42 M6

```
M6
```

Definition at line 932 of file MaterialModels.py.

7.48.4.43 pinchx

```
pinchx
```

Definition at line 870 of file MaterialModels.py.

7.48.4.44 pinchy

```
pinchy
```

Definition at line 871 of file MaterialModels.py.

7.48.4.45 Ry

```
Ry
```

Definition at line 876 of file MaterialModels.py.

7.48.4.46 t_dp

```
t_dp
```

Definition at line 868 of file MaterialModels.py.

7.48.4.47 t_fbp

`t_fbp`

Definition at line 879 of file MaterialModels.py.

7.48.4.48 t_p

`t_p`

Definition at line 867 of file MaterialModels.py.

7.48.4.49 t_pz

`t_pz`

Definition at line 900 of file MaterialModels.py.

7.48.4.50 tf_b

`tf_b`

Definition at line 864 of file MaterialModels.py.

7.48.4.51 tf_c

`tf_c`

Definition at line 861 of file MaterialModels.py.

7.48.4.52 V1

`V1`

Definition at line 926 of file MaterialModels.py.

7.48.4.53 V4

`V4`

Definition at line 927 of file MaterialModels.py.

7.48.4.54 V6

`V6`

Definition at line 928 of file MaterialModels.py.

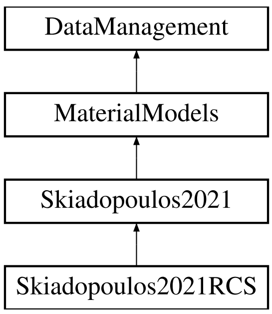
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.49 Skiadopoulos2021RCS Class Reference

WIP: Class that is the children of Skiadopoulos2021 and it's used for the panel zone spring in a RCS (RC column continous, Steel beam).

Inheritance diagram for Skiadopoulos2021RCS:



Public Member Functions

- def __init__ (self, int ID, SteelIShape beam, d_col, t_fbp=0, t_dp=0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0, dmg2=0, beta=0, safety_factor=False)
  Constructor of the class.

Public Attributes

- beam
- beam_section_name_tag

Additional Inherited Members

7.49.1 Detailed Description

WIP: Class that is the children of Skiadopoulos2021 and it's used for the panel zone spring in a RCS (RC column continous, Steel beam).

Parameters

Skiadopoulos2021	Parent class.

Definition at line 1084 of file MaterialModels.py.

7.49.2 Constructor & Destructor Documentation

7.49.2.1 __init__()

```
def __init__ (
              self,
             int ID,
             SteelIShape beam,
              d_col,
              t_fbp = 0,
              t_dp = 0,
              a_s = 0.03,
              pinchx = 0.25,
              pinchy = 0.75,
              dmg1 = 0,
              dmg2 = 0,
              beta = 0,
              safety_factor = False )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class SteelIShape. The copy of the section (beam) passed is stored in the member variable self.beam.

Parameters

ID	(int): Unique material model ID.
beam	(SteelIShape): SteelIShape beam section object.
d_col	(float): Depth of the RC column (continous)
t_fbp	(float, optional): Thickness of the face bearing plate (if present). Defaults to 0.
t_dp	(float, optional): Doubler plate thickness. Defaults to 0.0.
a_s	(float, optional): Strain hardening. Defaults to 0.03.
pinchx	(float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25
pinchy	(float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75
dmg1	(float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
dmg2	(float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
beta	(float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
safety_factor	(bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.

Reimplemented from Skiadopoulos2021.

Definition at line 1090 of file MaterialModels.py.

```
01091         t_dp=0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0, dmg2=0, beta=0, safety_factor=False):
01092         """
01093         Constructor of the class. It passes the arguments into the parent class to generate the combination of the parent class
01094             and the section class SteelIShape.
01095         The copy of the section (beam) passed is stored in the member variable self.beam.
01096 
01097         @param ID (int): Unique material model ID.
01098         @param beam (SteelIShape): SteelIShape beam section object.
01099         @param d_col (float): Depth of the RC column (continous)
01100         @param t_fbp (float, optional): Thickness of the face bearing plate (if present). Defaults to 0.
01101         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
01102         @param a_s (float, optional): Strain hardening. Defaults to 0.03.
01103         @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25
01104         @param pinchy (float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75
01105         @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
01106         @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
01107         @param beta (float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
01108         @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.
01109         """
01110         self.beam = deepcopy(beam)
01111         super().__init__(ID, beam.d, beam.bf, beam.tf, beam.Iy, d_col, 0, beam.Fy_web, beam.E, beam.tw,
01112             t_dp=t_dp, a_s=a_s, pinchx=pinchx, pinchy=pinchy, dmg1=dmg1, dmg2=dmg2, beta=beta, safety_factor=safety_factor, t_fbp=t_fbp)
01113         self.beam_section_name_tag = beam.name_tag
01114         self.UpdateStoredData()
01115 
01116 
```

7.49.3 Member Data Documentation

7.49.3.1 beam

`beam`

Definition at line 1110 of file MaterialModels.py.

7.49.3.2 beam_section_name_tag

`beam_section_name_tag`

Definition at line 1113 of file MaterialModels.py.

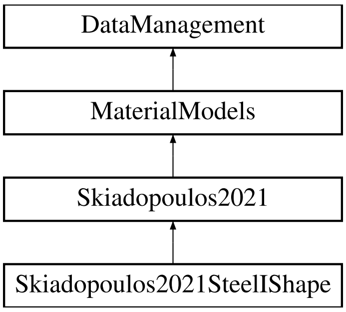
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.50 Skiadopoulos2021SteelIShape Class Reference

Class that is the children of Skiadopoulos2021 and combine the class SteelIShape (section) to retrieve the information needed.

Inheritance diagram for Skiadopoulos2021SteelIShape:



Public Member Functions

- def __init__ (self, int ID, SteelIShape col, SteelIShape beam, t_dp=0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0, dmg2=0, beta=0, safety_factor=False, t_fbp=0)

    Constructor of the class.

Public Attributes

- beam
- beam_section_name_tag
- col
- col_section_name_tag

Additional Inherited Members

7.50.1 Detailed Description

Class that is the children of Skiadopoulos2021 and combine the class SteelIShape (section) to retrieve the information needed.

Parameters

Skiadopoulos2021	Parent class.

Definition at line 1049 of file MaterialModels.py.

7.50.2 Constructor & Destructor Documentation

7.50.2.1 __init__()

```
def __init__ (
                self,
               int ID,
               SteelIShape col,
               SteelIShape beam,
                t_dp = 0,
                a_s = 0.03,
                pinchx = 0.25,
                pinchy = 0.75,
                dmg1 = 0,
                dmg2 = 0,
                beta = 0,
                safety_factor = False,
                t_fbp = 0 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class SteelIShape. The copy of the sections (col and beam) passed are stored in the member variable self.col and self.beam.

Parameters

ID	(int): Unique material model ID.
col	(SteelIShape): SteelIShape column section object.
beam	(SteelIShape): SteelIShape beam section object.
t_dp	(float, optional): Doubler plate thickness. Defaults to 0.0.
a_s	(float, optional): Strain hardening. Defaults to 0.03.
pinchx	(float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25.
pinchy	(float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75.
dmg1	(float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
dmg2	(float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
beta	(float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
safety_factor	(bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.
t_fbp	(float, optional): Thickness of the face bearing plate (if present). Defaults to 0.

Reimplemented from Skiadopoulos2021.

Definition at line 1055 of file MaterialModels.py.

```
01056         t_dp=0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0, dmg2=0, beta=0, safety_factor=False, t_fbp
      = 0):
01057         """
01058         Constructor of the class. It passes the arguments into the parent class to generate the
      combination of the parent class
01059             and the section class SteelIShape.
01060         The copy of the sections (col and beam) passed are stored in the member variable self.col and
      self.beam.
```

```
01061
01062         @param ID (int): Unique material model ID.
01063         @param col (SteelIShape): SteelIShape column section object.
01064         @param beam (SteelIShape): SteelIShape beam section object.
01065         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
01066         @param a_s (float, optional): Strain hardening. Defaults to 0.03.
01067         @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25.
01068         @param pinchy (float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75.
01069         @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
01070         @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
01071         @param beta (float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
01072         @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.
01073         @param t_fbp (float, optional): Thickness of the face bearing plate (if present). Defaults to 0.
01074         """
01075         self.col = deepcopy(col)
01076         self.beam = deepcopy(beam)
01077         super().__init__(ID, col.d, col.bf, col.tf, col.Iy, beam.d, beam.tf, col.Fy_web, col.E, col.tw,
01078             t_dp=t_dp, a_s=a_s, pinchx=pinchx, pinchy=pinchy, dmg1=dmg1, dmg2=dmg2, beta=beta, safety_factor=safety_factor, t_fbp=t_fbp)
01079         self.beam_section_name_tag = beam.name_tag
01080         self.col_section_name_tag = col.name_tag
01081         self.UpdateStoredData()
01082
01083
```

7.50.3 Member Data Documentation

7.50.3.1 beam

`beam`

Definition at line 1076 of file MaterialModels.py.

7.50.3.2 beam_section_name_tag

`beam_section_name_tag`

Definition at line 1079 of file MaterialModels.py.

7.50.3.3 col

`col`

Definition at line 1075 of file MaterialModels.py.

7.50.3.4 col_section_name_tag

```
col_section_name_tag
```

Definition at line 1080 of file MaterialModels.py.

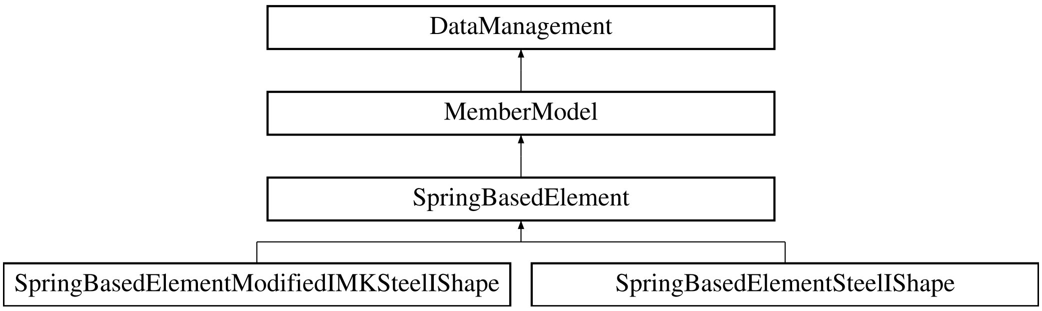
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.51 SpringBasedElement Class Reference

Class that handles the storage and manipulation of a spring-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Inheritance diagram for SpringBasedElement:



Public Member Functions

- def __init__ (self, int iNode_ID, int jNode_ID, A, E, Iy_mod, int geo_transf_ID, mat_ID_i=-1, mat_ID_j=-1, ele_ID=-1)

  Constructor of the class.
- def CreateMember (self)

  Method that initialises the member by calling the OpenSeesPy commands through various functions.
- def Record (self, str spring_or_element, str name_txt, str data_dir, force_rec=True, def_rec=True, time_↩ rec=True)

  Implementation of the homonym abstract method.
- def RecordNodeDef (self, str name_txt, str data_dir, time_rec=True)

  Implementation of the homonym abstract method.
- def ReInit (self, ele_ID=-1)

  Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False)

  Implementation of the homonym abstract method.
- def UpdateStoredData (self)

  Implementation of the homonym abstract method.

Public Attributes

- A
- data
- E
- ele_orientation
- element_array
- element_ID
- geo_transf_ID
- Initialized
- iNode_ID
- iNode_ID_spring
- iSpring_ID
- Iy_mod
- jNode_ID
- jNode_ID_spring
- jSpring_ID
- mat_ID_i
- mat_ID_j
- section_name_tag

7.51.1 Detailed Description

Class that handles the storage and manipulation of a spring-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Parameters

MemberModel	Parent abstract class.

Definition at line 659 of file MemberModel.py.

7.51.2 Constructor & Destructor Documentation

7.51.2.1 __init__()

```
def __init__ (
              self,
             int iNode_ID,
             int jNode_ID,
              A,
              E,
              Iy_mod,
             int geo_transf_ID,
              mat_ID_i = -1,
              mat_ID_j = -1,
              ele_ID = -1 )
```

Constructor of the class.

Parameters

iNode_ID	(int): ID of the first end node.
jNode_ID	(int): ID of the second end node.
A	(float): Area of the member.
E	(float): Young modulus.
Iy_mod	(float): Second moment of inertia (strong axis).
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
mat_ID_i	(int, optional): ID of the material model for the spring in the node i (if present). Defaults to -1.
mat_ID_j	(int, optional): ID of the material model for the spring in the node j (if present). Defaults to -1.
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	ID needs to be a positive integer.
NegativeValue	A needs to be positive.
NegativeValue	E needs to be positive.
NegativeValue	Iy_mod needs to be positive.
NegativeValue	ID needs to be a positive integer.
NegativeValue	ID needs to be a positive integer, if different from -1.
NegativeValue	ID needs to be a positive integer, if different from -1.
NameError	at least one spring needs to be defined.
NegativeValue	ID needs to be a positive integer, if different from -1.

Reimplemented in SpringBasedElementSteelIShape, and SpringBasedElementModifiedIMKSteelIShape.

Definition at line 665 of file MemberModel.py.

```
00665     def __init__(self, iNode_ID: int, jNode_ID: int, A, E, Iy_mod, geo_transf_ID: int, mat_ID_i = -1,
       mat_ID_j = -1, ele_ID = -1):
00666         """
00667         Constructor of the class.
00668
00669         @param iNode_ID (int): ID of the first end node.
00670         @param jNode_ID (int): ID of the second end node.
00671         @param A (float): Area of the member.
00672         @param E (float): Young modulus.
00673         @param Iy_mod (float): Second moment of inertia (strong axis).
00674         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
       documentation).
00675         @param mat_ID_i (int, optional): ID of the material model for the spring in the node i (if
       present). Defaults to -1.
00676         @param mat_ID_j (int, optional): ID of the material model for the spring in the node j (if
       present). Defaults to -1.
00677         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
       IDConvention to define it.
00678
00679         @exception NegativeValue: ID needs to be a positive integer.
00680         @exception NegativeValue: ID needs to be a positive integer.
00681         @exception NegativeValue: A needs to be positive.
00682         @exception NegativeValue: E needs to be positive.
00683         @exception NegativeValue: Iy_mod needs to be positive.
00684         @exception NegativeValue: ID needs to be a positive integer.
00685         @exception NegativeValue: ID needs to be a positive integer, if different from -1.
00686         @exception NegativeValue: ID needs to be a positive integer, if different from -1.
00687         @exception NameError: at least one spring needs to be defined.
00688         @exception NegativeValue: ID needs to be a positive integer, if different from -1.
00689         """
00690         # Check
00691         if iNode_ID < 1: raise NegativeValue()
00692         if jNode_ID < 1: raise NegativeValue()
00693         if A < 0: raise NegativeValue()
00694         if E < 0: raise NegativeValue()
00695         if Iy_mod < 0: raise NegativeValue()
```

```
        if geo_transf_ID < 1: raise NegativeValue()
        if mat_ID_i != -1 and mat_ID_i < 0: raise NegativeValue()
        if mat_ID_j != -1 and mat_ID_j < 0: raise NegativeValue()
        if mat_ID_i == -1 and mat_ID_j == -1: raise NameError("No springs defined for element ID = {}".format(IDConvention(iNode_ID, jNode_ID)))
        if ele_ID != -1 and ele_ID < 0: raise NegativeValue()

        # Arguments
        self.iNode_ID = iNode_ID
        self.jNode_ID = jNode_ID
        self.A = A
        self.E = E
        self.Iy_mod = Iy_mod
        self.geo_transf_ID = geo_transf_ID
        self.mat_ID_i = mat_ID_i
        self.mat_ID_j = mat_ID_j

        # Initialized the parameters that are dependent from others
        self.section_name_tag = "None"
        self.Initialized = False
        self.ReInit(ele_ID)


```

7.51.3 Member Function Documentation

7.51.3.1 CreateMember()

```
def CreateMember (
            self )
```

Method that initialises the member by calling the OpenSeesPy commands through various functions.

Definition at line 799 of file MemberModel.py.

```
00799     def CreateMember(self):
00800         """
00801         Method that initialises the member by calling the OpenSeesPy commands through various functions.
00802         """
00803         self.element_array = [[self.element_ID, self.iNode_ID_spring, self.jNode_ID_spring]]
00804         if self.mat_ID_i != -1:
00805             # Define zero length element i
00806             node(self.iNode_ID_spring, *nodeCoord(self.iNode_ID))
00807             self.iSpring_ID = IDConvention(self.iNode_ID, self.iNode_ID_spring)
00808             RotationalSpring(self.iSpring_ID, self.iNode_ID, self.iNode_ID_spring, self.mat_ID_i)
00809             self.element_array.append([self.iSpring_ID, self.iNode_ID, self.iNode_ID_spring])
00810         if self.mat_ID_j != -1:
00811             # Define zero length element j
00812             node(self.jNode_ID_spring, *nodeCoord(self.jNode_ID))
00813             self.jSpring_ID = IDConvention(self.jNode_ID, self.jNode_ID_spring)
00814             RotationalSpring(self.jSpring_ID, self.jNode_ID, self.jNode_ID_spring, self.mat_ID_j)
00815             self.element_array.append([self.jSpring_ID, self.jNode_ID, self.jNode_ID_spring])
00816
00817         # Define element
00818         element("elasticBeamColumn", self.element_ID, self.iNode_ID_spring, self.jNode_ID_spring, self.A, self.E, self.Iy_mod, self.geo_transf_ID)
00819
00820         # Update class
00821         self.Initialized = True
00822         self.UpdateStoredData()
00823
00824
```

7.51.3.2 Record()

```
def Record (
        self,
        str spring_or_element,
        str name_txt,
        str data_dir,
        force_rec = True,
        def_rec = True,
        time_rec = True )
```

Implementation of the homonym abstract method.

See parent class MemberModel for detailed information.

Reimplemented from MemberModel.

Definition at line 825 of file MemberModel.py.

```
00825     def Record(self, spring_or_element: str, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
00826         """
00827         Implementation of the homonym abstract method.
00828         See parent class MemberModel for detailed information.
00829         """
00830         if spring_or_element == "element":
00831             super().Record(self.element_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec, time_rec=time_rec)
00832         elif spring_or_element == "spring_i":
00833             if self.mat_ID_i == -1:
00834                 print("Spring i recorded not present in element ID = {}".format(self.element_ID))
00835             else:
00836                 super().Record(self.iSpring_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec, time_rec=time_rec)
00837         elif spring_or_element == "spring_j":
00838             if self.mat_ID_j == -1:
00839                 print("Spring j recorded not present in element ID = {}".format(self.element_ID))
00840             else:
00841                 super().Record(self.jSpring_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec, time_rec=time_rec)
00842         else:
00843             print("No recording option with: '{}' with element ID: {}".format(spring_or_element, self.element_ID))
00844
00845
```

7.51.3.3 RecordNodeDef()

```
def RecordNodeDef (
        self,
        str name_txt,
        str data_dir,
        time_rec = True )
```

Implementation of the homonym abstract method.

See parent class MemberModel for detailed information.

Reimplemented from MemberModel.

Definition at line 846 of file MemberModel.py.

```
00846     def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
00847         """
00848         Implementation of the homonym abstract method.
00849         See parent class MemberModel for detailed information.
00850         """
00851         super().RecordNodeDef(self.iNode_ID, self.jNode_ID, name_txt, data_dir, time_rec=time_rec)
00852
00853
```

7.51.3.4 ReInit()

```
def ReInit (
        self,
        ele_ID = -1 )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Definition at line 718 of file MemberModel.py.

```
    def ReInit(self, ele_ID = -1):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
        """
        # Members
        if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + " (modified)"
        # orientation:
        self.ele_orientation = NodesOrientation(self.iNode_ID, self.jNode_ID)
        if self.ele_orientation == "zero_length": raise ZeroLength(IDConvention(self.iNode_ID, self.jNode_ID))

        if self.mat_ID_i != -1:
            self.iNode_ID_spring = OffsetNodeIDConvention(self.iNode_ID, self.ele_orientation, "i")
        else:
            self.iNode_ID_spring = self.iNode_ID

        if self.mat_ID_j != -1:
            self.jNode_ID_spring = OffsetNodeIDConvention(self.jNode_ID, self.ele_orientation, "j")
        else:
            self.jNode_ID_spring = self.jNode_ID

        # element ID
        self.element_ID = IDConvention(self.iNode_ID_spring, self.jNode_ID_spring) if ele_ID == -1 else ele_ID

        # Data storage for loading/saving
        self.UpdateStoredData()


```

7.51.3.5 ShowInfo()

```
def ShowInfo (
        self,
        plot = False,
        block = False )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the material model. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 771 of file MemberModel.py.

```
00771     def ShowInfo(self, plot = False, block = False):
00772         """
00773         Implementation of the homonym abstract method.
00774         See parent class DataManagement for detailed information.
00775 
00776         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
      False.
00777         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
      of the program everytime that a plot should pop up). Defaults to False.
00778         """
00779         print("")
00780         print("Requested info for SpringBasedElement member model, ID = {}".format(self.element_ID))
00781         print("Section associated {} ".format(self.section_name_tag))
00782         print("Material model of the spring i, ID = {}".format(self.mat_ID_i))
00783         print("Material model of the spring j, ID = {}".format(self.mat_ID_j))
00784         print("Area A = {} mm2".format(self.A/mm2_unit))
00785         print("Young modulus E = {} GPa".format(self.E/GPa_unit))
00786         print("n modified moment of inertia Iy_mod = {} mm4".format(self.Iy_mod/mm4_unit))
00787         print("Geometric transformation = {}".format(self.geo_transf_ID))
00788         print("")
00789 
00790         if plot:
00791             if self.Initialized:
00792                 plot_member(self.element_array, "SpringBased Element, ID =
      {}".format(self.element_ID))
00793                 if block:
00794                     plt.show()
00795             else:
00796                 print("The SpringBasedElement is not initialized (node and elements not created), ID =
      {}".format(self.element_ID))
00797 
00798 
```

7.51.3.6 UpdateStoredData()

```
def UpdateStoredData (
                self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 749 of file MemberModel.py.

```
00749     def UpdateStoredData(self):
00750         """
00751         Implementation of the homonym abstract method.
00752         See parent class DataManagement for detailed information.
00753         """
00754         self.data = [["INFO_TYPE", "SpringBasedElement"], # Tag for differentiating different data
00755             ["element_ID", self.element_ID],
00756             ["section_name_tag", self.section_name_tag],
00757             ["A", self.A],
00758             ["E", self.E],
00759             ["Iy_mod", self.Iy_mod],
00760             ["iNode_ID", self.iNode_ID],
00761             ["iNode_ID_spring", self.iNode_ID_spring],
00762             ["mat_ID_i", self.mat_ID_i],
00763             ["jNode_ID", self.jNode_ID],
00764             ["jNode_ID_spring", self.jNode_ID_spring],
00765             ["mat_ID_j", self.mat_ID_j],
00766             ["ele_orientation", self.ele_orientation],
00767             ["tranf_ID", self.geo_transf_ID],
00768             ["Initialized", self.Initialized]]
00769 
00770 
```

7.51.4 Member Data Documentation

7.51.4.1 A

```
A
```

Definition at line 705 of file MemberModel.py.

7.51.4.2 data

```
data
```

Definition at line 754 of file MemberModel.py.

7.51.4.3 E

```
E
```

Definition at line 706 of file MemberModel.py.

7.51.4.4 ele_orientation

```
ele_orientation
```

Definition at line 728 of file MemberModel.py.

7.51.4.5 element_array

```
element_array
```

Definition at line 803 of file MemberModel.py.

7.51.4.6 element_ID

```
element_ID
```

Definition at line 742 of file MemberModel.py.

7.51.4.7 geo_transf_ID

geo_transf_ID

Definition at line 708 of file MemberModel.py.

7.51.4.8 Initialized

Initialized

Definition at line 714 of file MemberModel.py.

7.51.4.9 iNode_ID

iNode_ID

Definition at line 703 of file MemberModel.py.

7.51.4.10 iNode_ID_spring

iNode_ID_spring

Definition at line 732 of file MemberModel.py.

7.51.4.11 iSpring_ID

iSpring_ID

Definition at line 807 of file MemberModel.py.

7.51.4.12 Iy_mod

Iy_mod

Definition at line 707 of file MemberModel.py.

7.51.4.13 jNode_ID

`jNode_ID`

Definition at line 704 of file MemberModel.py.

7.51.4.14 jNode_ID_spring

`jNode_ID_spring`

Definition at line 737 of file MemberModel.py.

7.51.4.15 jSpring_ID

`jSpring_ID`

Definition at line 813 of file MemberModel.py.

7.51.4.16 mat_ID_i

`mat_ID_i`

Definition at line 709 of file MemberModel.py.

7.51.4.17 mat_ID_j

`mat_ID_j`

Definition at line 710 of file MemberModel.py.

7.51.4.18 section_name_tag

`section_name_tag`

Definition at line 713 of file MemberModel.py.

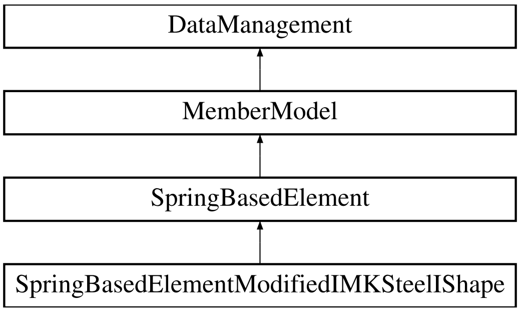
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.52 SpringBasedElementModifiedIMKSteelIShape Class Reference

Class that is the children of SpringBasedElement and combine the class SteelIShape (section) to retrieve the information needed.

Inheritance diagram for SpringBasedElementModifiedIMKSteelIShape:



Public Member Functions

- def __init__ (self, int iNode_ID, int jNode_ID, SteelIShape section, int geo_transf_ID, new_mat_ID_i=-1, new_mat_ID_j=-1, N_G=0, L_0=-1, L_b=-1, ele_ID=-1)

  Constructor of the class.

Public Attributes

- section
- section_name_tag

7.52.1 Detailed Description

Class that is the children of SpringBasedElement and combine the class SteelIShape (section) to retrieve the information needed.

If there are two springs and the inflection point not in the middle, use two spring elements, connect them rigidly in the inflection point with one spring each in the extremes. L_b is assumed the same for top and bottom springs.

Parameters

SpringBasedElement	Parent class.

Definition at line 891 of file MemberModel.py.

7.52.2 Constructor & Destructor Documentation

7.52.2.1 __init__()

```
def __init__ (
                self,
              int iNode_ID,
              int jNode_ID,
              SteelIShape section,
              int geo_transf_ID,
                new_mat_ID_i = -1,
                new_mat_ID_j = -1,
                N_G = 0,
                L_0 = -1,
                L_b = -1,
                ele_ID = -1 )
```

Constructor of the class.

Parameters

iNode_ID	(int): ID of the first end node.
jNode_ID	(int): ID of the second end node.
section	(SteelIShape): SteelIShape section object.
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
new_mat_↩ ID_i	(int, optional): New ID for the definition of the material model for the spring in the node i. If -1 is passed, the class generate no material model and no spring. If 0 is passed, no i spring. Defaults to -1.
new_mat_↩ ID_j	(int, optional): New ID for the definition of the material model for the spring in the node j. If -1 is passed, the class generate no material model and no spring. If 0 is passed, no j spring. Defaults to -1.
N_G	(float, optional): Axial load. Defaults to 0.
L_0	(float, optional): Distance from the maximal moment to zero. Defaults to -1, e.g. computed in **init**().
L_b	(float, optional): Maximal unbraced lateral buckling length. Defaults to -1, e.g. computed in **init**().
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	ID needs to be a positive integer.
NameError	at least one spring needs to be defined.
NegativeValue	ID needs to be a positive integer.
ZeroLength	The two nodes are superimposed.

Reimplemented from SpringBasedElement.

Definition at line 899 of file MemberModel.py.

```
00900             N_G = 0, L_0 = -1, L_b = -1, ele_ID = -1):
00901             """
00902             Constructor of the class.
00903 
00904             @param iNode_ID (int): ID of the first end node.
00905             @param jNode_ID (int): ID of the second end node.
00906             @param section (SteelIShape): SteelIShape section object.
00907             @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
      documentation).
```

```
00908         @param new_mat_ID_i (int, optional): New ID for the definition of the material model for the spring in the node i.
00909             If -1 is passed, the class generate no material model and no spring. If 0 is passed, no i spring. Defaults to -1.
00910         @param new_mat_ID_j (int, optional): New ID for the definition of the material model for the spring in the node j.
00911             If -1 is passed, the class generate no material model and no spring. If 0 is passed, no j spring. Defaults to -1.
00912         @param N_G (float, optional): Axial load. Defaults to 0.
00913         @param L_0 (float, optional): Distance from the maximal moment to zero. Defaults to -1, e.g. computed in __init__().
00914         @param L_b (float, optional): Maximal unbraced lateral buckling length. Defaults to -1, e.g. computed in __init__().
00915         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
00916
00917         @exception NegativeValue: ID needs to be a positive integer.
00918         @exception NegativeValue: ID needs to be a positive integer.
00919         @exception NameError: at least one spring needs to be defined.
00920         @exception NegativeValue: ID needs to be a positive integer.
00921         @exception ZeroLength: The two nodes are superimposed.
00922         """
00923         self.section = deepcopy(section)
00924         if new_mat_ID_i != -1 and new_mat_ID_i < 0: raise NegativeValue()
00925         if new_mat_ID_j != -1 and new_mat_ID_j < 0: raise NegativeValue()
00926         if new_mat_ID_i == 0 and new_mat_ID_j == 0: raise NameError("No springs imposed for element ID = {}. Use ElasticElement instead".format(IDConvention(iNode_ID, jNode_ID)))
00927         if ele_ID != -1 and ele_ID < 0: raise NegativeValue()
00928
00929         if L_0 == -1:
00930             if new_mat_ID_i != 0 and new_mat_ID_j != 0:
00931                 L_0 = section.L/2
00932             else:
00933                 L_0 = section.L
00934         L_b = L_0 if L_b == -1 else L_b
00935
00936         # auto assign ID for material of springs
00937         ele_orientation = NodesOrientation(iNode_ID, jNode_ID)
00938         if ele_orientation == "zero_length": raise ZeroLength(IDConvention(iNode_ID, jNode_ID))
00939         if new_mat_ID_i != 0 and new_mat_ID_i == -1:
00940             new_mat_ID_i = OffsetNodeIDConvention(iNode_ID, ele_orientation, "i")
00941         if new_mat_ID_j != 0 and new_mat_ID_j == -1:
00942             new_mat_ID_j = OffsetNodeIDConvention(jNode_ID, ele_orientation, "j")
00943
00944         if new_mat_ID_i != 0:
00945             # Create mat i
00946             iSpring = ModifiedIMKSteelIShape(new_mat_ID_i, section, N_G, L_0 = L_0, L_b = L_b)
00947             iSpring.Bilin()
00948
00949         if new_mat_ID_j != 0:
00950             # Create mat j
00951             jSpring = ModifiedIMKSteelIShape(new_mat_ID_j, section, N_G, L_0 = L_0, L_b = L_b)
00952             jSpring.Bilin()
00953
00954         new_mat_ID_i = -1 if new_mat_ID_i == 0 else new_mat_ID_i
00955         new_mat_ID_j = -1 if new_mat_ID_j == 0 else new_mat_ID_j
00956
00957         super().__init__(iNode_ID, jNode_ID, section.A, section.E, section.Iy_mod, geo_transf_ID, mat_ID_i=new_mat_ID_i, mat_ID_j=new_mat_ID_j, ele_ID=ele_ID)
00958         self.section_name_tag = section.name_tag
00959         self.UpdateStoredData()
00960         # Check length
00961         self._CheckL()
00962
00963
```

7.52.3 Member Data Documentation

7.52.3.1 section

`section`

Definition at line 923 of file MemberModel.py.

7.52.3.2 section_name_tag

```
section_name_tag
```

Definition at line 958 of file MemberModel.py.

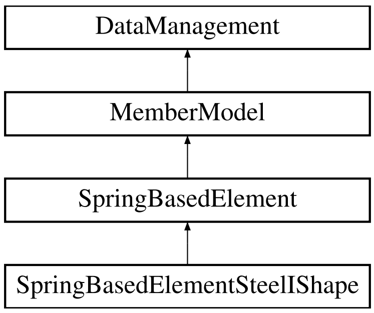
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.53 SpringBasedElementSteelIShape Class Reference

Class that is the children of SpringBasedElement and combine the class SteelIShape (section) to retrieve the information needed.

Inheritance diagram for SpringBasedElementSteelIShape:



Public Member Functions

- def __init__ (self, int iNode_ID, int jNode_ID, SteelIShape section, int geo_transf_ID, mat_ID_i=-1, mat_ID_j=-1, ele_ID=-1)

    Constructor of the class.

Public Attributes

- section
- section_name_tag

7.53.1 Detailed Description

Class that is the children of SpringBasedElement and combine the class SteelIShape (section) to retrieve the information needed.

L_b is assumed the same for top and bottom springs.

Parameters

SpringBasedElement	Parent class.

Definition at line 854 of file MemberModel.py.

7.53.2 Constructor & Destructor Documentation

7.53.2.1 __init__()

```
def __init__ (
                self,
               int iNode_ID,
               int jNode_ID,
               SteelIShape section,
               int geo_transf_ID,
                mat_ID_i = -1,
                mat_ID_j = -1,
                ele_ID = -1 )
```

Constructor of the class.

Parameters

iNode_ID	(int): ID of the first end node.
jNode_ID	(int): ID of the second end node.
section	(SteelIShape): SteelIShape section object.
geo_transf_ID	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
mat_ID_i	(int, optional): ID of the material model for the spring in the node i (if present). Defaults to -1.
mat_ID_j	(int, optional): ID of the material model for the spring in the node j (if present). Defaults to -1.
ele_ID	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	ID needs to be a positive integer.
NameError	at least one spring needs to be defined.
NegativeValue	ID needs to be a positive integer.

Reimplemented from SpringBasedElement.

Definition at line 861 of file MemberModel.py.

```
00861     def __init__(self, iNode_ID: int, jNode_ID: int, section: SteelIShape, geo_transf_ID: int,
       mat_ID_i=-1, mat_ID_j=-1, ele_ID = -1):
00862         """
00863         Constructor of the class.
00864 
00865         @param iNode_ID (int): ID of the first end node.
00866         @param jNode_ID (int): ID of the second end node.
```

```
00867         @param section (SteelIShape): SteelIShape section object.
00868         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
     documentation).
00869         @param mat_ID_i (int, optional): ID of the material model for the spring in the node i (if
     present). Defaults to -1.
00870         @param mat_ID_j (int, optional): ID of the material model for the spring in the node j (if
     present). Defaults to -1.
00871         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
     IDConvention to define it.
00872
00873         @exception NegativeValue: ID needs to be a positive integer.
00874         @exception NegativeValue: ID needs to be a positive integer.
00875         @exception NameError: at least one spring needs to be defined.
00876         @exception NegativeValue: ID needs to be a positive integer.
00877         """
00878         self.section = deepcopy(section)
00879         if mat_ID_i != -1 and mat_ID_i < 0: raise NegativeValue()
00880         if mat_ID_j != -1 and mat_ID_j < 0: raise NegativeValue()
00881         if mat_ID_i == -1 and mat_ID_j == -1: raise NameError("No springs defined for element ID =
     {}".format(IDConvention(iNode_ID, jNode_ID)))
00882         if ele_ID != -1 and ele_ID < 0: raise NegativeValue()
00883
00884         super().__init__(iNode_ID, jNode_ID, section.A, section.E, section.Iy_mod, geo_transf_ID,
     mat_ID_i=mat_ID_i, mat_ID_j=mat_ID_j, ele_ID=ele_ID)
00885         self.section_name_tag = section.name_tag
00886         self.UpdateStoredData()
00887         # Check length
00888         self._CheckL()
00889
00890
```

7.53.3 Member Data Documentation

7.53.3.1 section

`section`

Definition at line 878 of file MemberModel.py.

7.53.3.2 section_name_tag

`section_name_tag`

Definition at line 885 of file MemberModel.py.

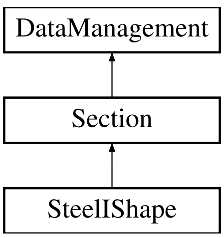
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py

7.54 SteelIShape Class Reference

Class that stores funcions, geometric and mechanical properties of a steel double symmetric I-shape profile.

Inheritance diagram for SteelIShape:



Public Member Functions

- def __init__ (self, str Type, d, bf, tf, tw, L, r, E, Fy, Fy_web=-1, name_tag="Not Defined")
  The conctructor of the class.
- def Compute_iy (self)
  Compute the gyration radius with respect to the strong axis.
- def Compute_iz (self)
  Compute the gyration radius with respect to the weak axis.
- def ComputeA (self)
  Compute the area of a double symmetric I-profile section with fillets.
- def ComputeIy (self)
  Compute the moment of inertia of a double symmetric I-profile section, with respect to its strong axis with fillets.
- def ComputeIz (self)
  Compute the moment of inertia of a double symmetric I-profile section, with respect to its weak axis with fillets.
- def ComputeWply (self)
  Compute the plastic modulus of a double symmetric I-profile section, with respect to its strong axis with fillets.
- def ComputeWplz (self)
  Compute the plastic modulus of a double symmetric I-profile section, with respect to its weak axis with fillets.
- def ReInit (self)
  Implementation of the homonym abstract method.
- def ShowInfo (self)
  Implementation of the homonym abstract method.
- def UpdateStoredData (self)
  Implementation of the homonym abstract method.

Public Attributes

- A
- bf
- d
- data
- E
- Fy
- Fy_web
- h_1
- Iy
- iy
- Iy_mod
- Iz
- iz
- L
- My
- name_tag
- Npl
- r
- tf
- tw
- Type
- Wply
- Wplz

Static Public Attributes

- float n = 10.0

7.54.1 Detailed Description

Class that stores funcions, geometric and mechanical properties of a steel double symmetric I-shape profile.

The parameter 'n' is used as global throughout the SteelIShape sections to optimise the program (given the fact that is constant everytime).

Parameters

Section	Parent abstract class.

Definition at line 22 of file Section.py.

7.54.2 Constructor & Destructor Documentation

7.54.2.1 __init__()

```
def __init__ (
              self,
             str Type,
              d,
              bf,
              tf,
              tw,
              L,
              r,
              E,
              Fy,
              Fy_web = -1,
              name_tag = "Not Defined" )
```

The conctructor of the class.

Parameters

Type	(str): Type of the section. It can be 'Col' for column or 'Beam' for beams.
d	(float): Depth of the section.
bf	(float): Flange's width of the section
tf	(float): Flange's thickness of the section
tw	(float): Web's thickness of the section
L	(float): Effective length of the element associated with this section. If the panel zone is present, exclude its dimension.
r	(float): Radius of the weld fillets of the section.
E	(float): Young modulus of the section.
Fy	(float): Yield strength of the flange of the section. Used as the yield strength of the entire section.
Fy_web	(float, optional): Yield strength of the web of the section. Used for panel zone associated to this section. Defaults to -1, e.g. computed in **init**() as equal to Fy.
name_tag	(str, optional): Name TAG of the section. Defaults to "Not Defined".

Exceptions

WrongArgument	Type needs to be 'Col' or 'Beam'.
NegativeValue	d needs to be positive.
NegativeValue	bf needs to be positive.
NegativeValue	tf needs to be positive.
NegativeValue	tw needs to be positive.
NegativeValue	L needs to be positive.
NegativeValue	r needs to be positive.
NegativeValue	E needs to be positive.
NegativeValue	Fy needs to be positive.
NegativeValue	Fy_web needs to be positive if different from -1.
InconsistentGeometry	tw should be smaller than bf.
InconsistentGeometry	tf needs to be smaller than half of d
InconsistentGeometry	r should be less than half bf and d

Definition at line 32 of file Section.py.

```
00032     def __init__(self, Type: str, d, bf, tf, tw, L, r, E, Fy, Fy_web = -1, name_tag = "Not Defined"):
00033         """
00034         The conctructor of the class.
00035 
00036         @param Type (str): Type of the section. It can be 'Col' for column or 'Beam' for beams.
00037         @param d (float): Depth of the section.
00038         @param bf (float): Flange's width of the section
00039         @param tf (float): Flange's thickness of the section
00040         @param tw (float): Web's thickness of the section
00041         @param L (float): Effective length of the element associated with this section.
00042             If the panel zone is present, exclude its dimension.
00043         @param r (float): Radius of the weld fillets of the section.
00044         @param E (float): Young modulus of the section.
00045         @param Fy (float): Yield strength of the flange of the section. Used as the yield strength of
       the entire section.
00046         @param Fy_web (float, optional): Yield strength of the web of the section. Used for panel zone
       associated to this section.
00047             Defaults to -1, e.g. computed in __init__() as equal to Fy.
00048         @param name_tag (str, optional): Name TAG of the section. Defaults to "Not Defined".
00049 
00050         @exception WrongArgument: Type needs to be 'Col' or 'Beam'.
00051         @exception NegativeValue: d needs to be positive.
00052         @exception NegativeValue: bf needs to be positive.
00053         @exception NegativeValue: tf needs to be positive.
00054         @exception NegativeValue: tw needs to be positive.
00055         @exception NegativeValue: L needs to be positive.
00056         @exception NegativeValue: r needs to be positive.
00057         @exception NegativeValue: E needs to be positive.
00058         @exception NegativeValue: Fy needs to be positive.
00059         @exception NegativeValue: Fy_web needs to be positive if different from -1.
00060         @exception InconsistentGeometry: tw should be smaller than bf.
00061         @exception InconsistentGeometry: tf needs to be smaller than half of d
00062         @exception InconsistentGeometry: r should be less than half bf and d
00063         """
00064         # Check
00065         if Type != "Beam" and Type != "Col": raise WrongArgument()
00066         if d < 0: raise NegativeValue()
00067         if bf < 0: raise NegativeValue()
00068         if tf < 0: raise NegativeValue()
00069         if tw < 0: raise NegativeValue()
00070         if L < 0: raise NegativeValue()
00071         if r < 0: raise NegativeValue()
00072         if E < 0: raise NegativeValue()
00073         if Fy < 0: raise NegativeValue()
00074         if Fy_web != -1 and Fy_web < 0: raise NegativeValue()
00075         if tw > bf: raise InconsistentGeometry()
00076         if tf > d/2: raise InconsistentGeometry()
00077         if r > bf/2 or r > d/2: raise InconsistentGeometry()
00078 
00079         # Arguments
00080         self.Type = Type
00081         self.d = d
00082         self.bf = bf
00083         self.tf = tf
00084         self.tw = tw
00085         self.L = L
```

```
00086         self.r = r
00087         self.E = E
00088         self.Fy = Fy
00089         self.Fy_web = Fy if Fy_web == -1 else Fy_web
00090         self.name_tag = name_tag
00091 
00092         # Initialized the parameters that are dependent from others
00093         self.ReInit()
00094 
```

7.54.3 Member Function Documentation

7.54.3.1 Compute_iy()

```
def Compute_iy (
              self )
```

Compute the gyration radius with respect to the strong axis.

Returns

float: The gyration radius with respect to the strong axis.

Definition at line 241 of file Section.py.

```
00241     def Compute_iy(self):
00242         """
00243         Compute the gyration radius with respect to the strong axis.
00244 
00245         @returns float: The gyration radius with respect to the strong axis.
00246         """
00247         #  Iy :     The second moment of inertia with respect to thte strong axis
00248         #  A :      The area
00249 
00250         return math.sqrt(self.Iy/self.A)
00251 
```

7.54.3.2 Compute_iz()

```
def Compute_iz (
              self )
```

Compute the gyration radius with respect to the weak axis.

Returns

float: The gyration radius with respect to the weak axis.

Definition at line 252 of file Section.py.

```
00252     def Compute_iz(self):
00253         """
00254         Compute the gyration radius with respect to the weak axis.
00255 
00256         @returns float: The gyration radius with respect to the weak axis.
00257         """
00258         #  Iz :     The second moment of inertia with respect to thte weak axis
00259         #  A :      The area
00260 
00261         return math.sqrt(self.Iz/self.A)
00262 
00263 
```

7.54.3.3 ComputeA()

```
def ComputeA (
              self )
```

Compute the area of a double symmetric I-profile section with fillets.

Returns

float: Area of the I shape section (with fillets included)

Definition at line 170 of file Section.py.

```
00170     def ComputeA(self):
00171         """
00172         Compute the area of a double symmetric I-profile section with fillets.
00173 
00174         @returns float: Area of the I shape section (with fillets included)
00175         """
00176         #  d :     The depth
00177         #  bf :    The flange's width
00178         #  tf :    The flange's thickness
00179         #  tw :    The web's thickness
00180         #  r :     The weld fillet radius
00181 
00182         # without fillets bf*tf*2 + tw*(d-2*tf)
00183         return 2.0*self.bf*self.tf+self.tw*(self.d-2.0*self.tf)+0.8584*self.r**2
00184 
```

7.54.3.4 ComputeIy()

```
def ComputeIy (
              self )
```

Compute the moment of inertia of a double symmetric I-profile section, with respect to its strong axis with fillets.

Returns

float: The moment of inertia with respect to the strong axis.

Definition at line 185 of file Section.py.

```
00185     def ComputeIy(self):
00186         """
00187         Compute the moment of inertia of a double symmetric I-profile section, with respect to its
       strong axis with fillets.
00188 
00189         @returns float: The moment of inertia with respect to the strong axis.
00190         """
00191         #  d :     The depth
00192         #  bf :    The flange's width
00193         #  tf :    The flange's thickness
00194         #  tw :    The web's thickness
00195         #  r :     The weld fillet radius
00196 
00197         # without fillets: bf*tf/2*(d-tf)**2 + bf*tf**3/6 + (d-tf*2)**3*tf/12
00198         return
       (self.bf*self.d**3.0-(self.bf-self.tw)*(self.d-2.0*self.tf)**3)/12.0+0.8584*self.r**2*(0.5*self.d-self.tf-0.4467*self.
00199 
```

7.54.3.5 ComputeIz()

```
def ComputeIz (
                self )
```

Compute the moment of inertia of a double symmetric I-profile section, with respect to its weak axis with fillets.

Returns

float: The moment of inertia with respect to the weak axis.

Definition at line 200 of file Section.py.

```
00200     def ComputeIz(self):
00201         """
00202         Compute the moment of inertia of a double symmetric I-profile section, with respect to its
      weak axis with fillets.
00203 
00204         @returns float: The moment of inertia with respect to the weak axis.
00205         """
00206         #  d :     The depth
00207         #  bf :    The flange's width
00208         #  tf :    The flange's thickness
00209         #  tw :    The web's thickness
00210         #  r :     The weld fillet radius
00211 
00212         return
      (self.tf*self.bf**3)/6.0+((self.d-2.0*self.tf)*self.tw**3)/12.0+0.8584*self.r**2*(0.5*self.tw+0.2234*self.r)**2
00213 
```

7.54.3.6 ComputeWply()

```
def ComputeWply (
                self )
```

Compute the plastic modulus of a double symmetric I-profile section, with respect to its strong axis with fillets.

Returns

float: The plastic modulus with respect to the strong axis.

Definition at line 214 of file Section.py.

```
00214     def ComputeWply(self):
00215         """
00216         Compute the plastic modulus of a double symmetric I-profile section, with respect to its
      strong axis with fillets.
00217 
00218         @returns float: The plastic modulus with respect to the strong axis.
00219         """
00220         #  d :     The depth
00221         #  bf :    The flange's width
00222         #  tf :    The flange's thickness
00223         #  tw :    The web's thickness
00224         #  r :     The weld fillet radius
00225 
00226         return
      self.bf*self.tf*(self.d-self.tf)+(self.d-2.0*self.tf)**2.0*(self.tw/4.0)+0.4292*self.r**2*(self.d-2.0*self.tf-0.4467*se
00227 
```

7.54.3.7 ComputeWplz()

```
def ComputeWplz (
                self )
```

Compute the plastic modulus of a double symmetric I-profile section, with respect to its weak axis with fillets.

Returns

float: The plastic modulus with respect to the weak axis.

Definition at line 228 of file Section.py.

```
    def ComputeWplz(self):
        """
        Compute the plastic modulus of a double symmetric I-profile section, with respect to its weak axis with fillets.

        @returns float: The plastic modulus with respect to the weak axis.
        """
        #  d :     The depth
        #  bf :    The flange's width
        #  tf :    The flange's thickness
        #  tw :    The web's thickness
        #  r :     The weld fillet radius
        return (self.tf*self.bf**2)/2+(self.d-2.0*self.tf)*(self.tw**2/4.0)+0.4292*self.r**2*(self.tw+0.4467*self.r)

```

7.54.3.8 ReInit()

```
def ReInit (
                self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 95 of file Section.py.

```
    def ReInit(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        # Member
        self.h_1 = self.d - 2.0*self.r -2.0*self.tf
        self.A = self.ComputeA()
        self.Npl = self.A*self.Fy
        self.Iy = self.ComputeIy()
        self.Iz = self.ComputeIz()
        self.Wply = self.ComputeWply()
        self.Wplz = self.ComputeWplz()
        self.My = self.Fy*self.Wply
        self.Iy_mod = self.Iy*(n + 1.0)/n
        self.iz = self.Compute_iz()
        self.iy = self.Compute_iy()

        # Data storage for loading/saving
        self.UpdateStoredData()

```

7.54.3.9 ShowInfo()

```
def ShowInfo (
              self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 145 of file Section.py.

```
00145     def ShowInfo(self):
00146         """
00147         Implementation of the homonym abstract method.
00148         See parent class DataManagement for detailed information.
00149         """
00150         print("")
00151         print("Requested info for steel I shape section of type = {} and name tag =
      {}".format(self.Type, self.name_tag))
00152         print("d = {} mm".format(self.d/mm_unit))
00153         print("Fy = {} MPa".format(self.Fy/MPa_unit))
00154         print("Fy web = {} MPa".format(self.Fy_web/MPa_unit))
00155         print("E = {} GPa".format(self.E/GPa_unit))
00156         print("h_1 = {} mm".format(self.h_1/mm_unit))
00157         print("A = {} mm2".format(self.A/mm2_unit))
00158         print("Iy = {} mm4".format(self.Iy/mm4_unit))
00159         print("Iz = {} mm4".format(self.Iz/mm4_unit))
00160         print("Wply = {} mm3".format(self.Wply/mm3_unit))
00161         print("Wplz = {} mm3".format(self.Wplz/mm3_unit))
00162         print("Iy_mod = {} mm4".format(self.Iy_mod/mm4_unit))
00163         print("iy = {} mm".format(self.iy/mm_unit))
00164         print("iz = {} mm".format(self.iz/mm_unit))
00165         print("My = {} kNm".format(self.My/kNm_unit))
00166         print("Npl = {} kN".format(self.Npl/kN_unit))
00167         print("")
00168 
00169 
```

7.54.3.10 UpdateStoredData()

```
def UpdateStoredData (
              self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 116 of file Section.py.

```
00116     def UpdateStoredData(self):
00117         """
00118         Implementation of the homonym abstract method.
00119         See parent class DataManagement for detailed information.
00120         """
00121         self.data = [["INFO_TYPE", "SteelIShape"], # Tag for differentiating different data
00122             ["name_tag", self.name_tag],
00123             ["Type", self.Type],
00124             ["d", self.d],
00125             ["bf", self.bf],
00126             ["tf", self.tf],
00127             ["tw", self.tw],
00128             ["L", self.L],
00129             ["r", self.r],
00130             ["h_1", self.h_1],
00131             ["E", self.E],
00132             ["Fy", self.Fy],
00133             ["Fy_web", self.Fy_web],
00134             ["A", self.A],
00135             ["Iy", self.Iy],
00136             ["Iz", self.Iz],
00137             ["Wply", self.Wply],
00138             ["Wplz", self.Wplz],
00139             ["Iy_mod", self.Iy_mod],
00140             ["iy", self.iy],
00141             ["iz", self.iz],
00142             ["Npl", self.Npl],
00143             ["My", self.My]]
00144 
```

7.54.4 Member Data Documentation

7.54.4.1 A

```
A
```

Definition at line 102 of file Section.py.

7.54.4.2 bf

```
bf
```

Definition at line 82 of file Section.py.

7.54.4.3 d

```
d
```

Definition at line 81 of file Section.py.

7.54.4.4 data

```
data
```

Definition at line 121 of file Section.py.

7.54.4.5 E

```
E
```

Definition at line 87 of file Section.py.

7.54.4.6 Fy

```
Fy
```

Definition at line 88 of file Section.py.

7.54.4.7 Fy_web

```
Fy_web
```

Definition at line 89 of file Section.py.

7.54.4.8 h_1

```
h_1
```

Definition at line 101 of file Section.py.

7.54.4.9 Iy

```
Iy
```

Definition at line 104 of file Section.py.

7.54.4.10 iy

```
iy
```

Definition at line 111 of file Section.py.

7.54.4.11 Iy_mod

```
Iy_mod
```

Definition at line 109 of file Section.py.

7.54.4.12 Iz

```
Iz
```

Definition at line 105 of file Section.py.

7.54.4.13 iz

```
iz
```

Definition at line 110 of file Section.py.

7.54.4.14 L

```
L
```

Definition at line 85 of file Section.py.

7.54.4.15 My

```
My
```

Definition at line 108 of file Section.py.

7.54.4.16 n

```
float n = 10.0  [static]
```

Definition at line 30 of file Section.py.

7.54.4.17 name_tag

```
name_tag
```

Definition at line 90 of file Section.py.

7.54.4.18 Npl

```
Npl
```

Definition at line 103 of file Section.py.

7.54.4.19 r

```
r
```

Definition at line 86 of file Section.py.

7.54.4.20 tf

```
tf
```

Definition at line 83 of file Section.py.

7.54.4.21 tw

```
tw
```

Definition at line 84 of file Section.py.

7.54.4.22 Type

```
Type
```

Definition at line 80 of file Section.py.

7.54.4.23 Wply

```
Wply
```

Definition at line 106 of file Section.py.

7.54.4.24 Wplz

`Wplz`

Definition at line 107 of file Section.py.

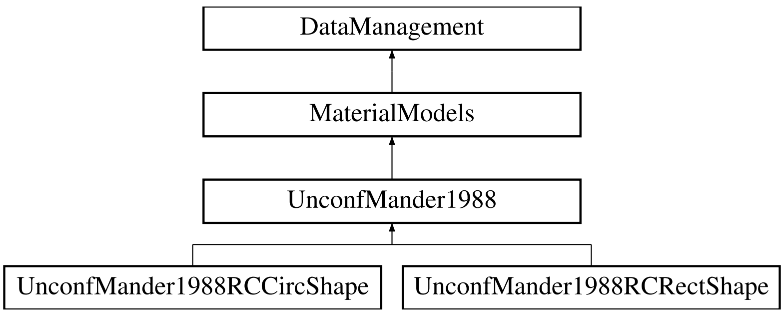
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Section.py

7.55 UnconfMander1988 Class Reference

Class that stores funcions and material properties of a RC rectangular or circular section with Mander 1988 as the material model for the unconfined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

Inheritance diagram for UnconfMander1988:



Public Member Functions

- def __init__ (self, int ID, fc, Ec, ec=1, ecp=1, fct=-1, et=-1, beta=0.1)

  Constructor of the class.
- def CheckApplicability (self)

  Implementation of the homonym abstract method.
- def Compute_ec (self)

  Method that computes the compressive concrete yield strain.
- def Compute_ecp (self)

  Method that computes the compressive concrete spalling strain.
- def Compute_ecu (self)

  Method that computes the compressive concrete failure strain.
- def Compute_et (self)

  Method that computes the tensile concrete yield strain.
- def Compute_fct (self)

  Method that computes the tensile concrete yield stress.
- def Concrete01 (self)

  Generate the material model Concrete01 for unconfined concrete using the computed parameters.
- def Concrete04 (self)

  Generate the material model Concrete04 for unconfined concrete (Mander 1988) using the computed parameters.
- def ReInit (self, ec=1, ecp=1, fct=-1, et=-1)

  Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False, concrete04=True)

  Implementation of the homonym abstract method.
- def UpdateStoredData (self)

  Implementation of the homonym abstract method.

Public Attributes

- beta
- data
- Ec
- ec
- ecp
- ecu
- et
- fc
- fct
- ID
- Initialized
- section_name_tag

7.55.1 Detailed Description

Class that stores funcions and material properties of a RC rectangular or circular section with Mander 1988 as the material model for the unconfined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

For more information about the empirical model for the computation of the parameters, see Mander et Al. 1988, Karthik and Mander 2011 and SIA 262:2012.

Parameters

MaterialModels	Parent abstract class.

Definition at line 1117 of file MaterialModels.py.

7.55.2 Constructor & Destructor Documentation

7.55.2.1 __init__()

```
def __init__ (
              self,
             int ID,
              fc,
              Ec,
              ec = 1,
              ecp = 1,
              fct = -1,
              et = -1,
              beta = 0.1 )
```

Constructor of the class.

Parameters

ID	(int): Unique material model ID.
fc	(float): Compressive concrete yield strength (needs to be negative).
Ec	(float): Young modulus.
ec	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
ecp	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
fct	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
et	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
beta	(float, optional): Loating point value defining the exponential curve parameter to define the residual stress. Defaults to 0.1 (according to OpenSeesPy documentation)

Exceptions

NegativeValue	ID needs to be a positive integer.
PositiveValue	fc needs to be negative.
NegativeValue	Ec needs to be positive.
PositiveValue	ec needs to be negative if different from 1.
PositiveValue	ecp needs to be positive if different from 1.
NegativeValue	fct needs to be positive if different from -1.
NegativeValue	et needs to be positive if different from -1.

Reimplemented in UnconfMander1988RCCircShape, and UnconfMander1988RCRectShape.

Definition at line 1125 of file MaterialModels.py.

```
01125     def __init__(self, ID: int, fc, Ec, ec = 1, ecp = 1, fct = -1, et = -1, beta = 0.1):
01126         """
01127         Constructor of the class.
01128
01129         @param ID (int): Unique material model ID.
01130         @param fc (float): Compressive concrete yield strength (needs to be negative).
01131         @param Ec (float): Young modulus.
01132         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
      according to Karthik and Mander 2011.
01133         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
      to Mander 1988.
01134         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
      according to SIA 262:2012.
01135         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
      according to SIA 262:2012.
01136         @param beta (float, optional): Loating point value defining the exponential curve parameter to
      define the residual stress.
01137             Defaults to 0.1 (according to OpenSeesPy documentation)
01138
01139         @exception NegativeValue: ID needs to be a positive integer.
01140         @exception PositiveValue: fc needs to be negative.
01141         @exception NegativeValue: Ec needs to be positive.
01142         @exception PositiveValue: ec needs to be negative if different from 1.
01143         @exception PositiveValue: ecp needs to be positive if different from 1.
01144         @exception NegativeValue: fct needs to be positive if different from -1.
01145         @exception NegativeValue: et needs to be positive if different from -1.
01146         """
01147         # Check
01148         if ID < 0: raise NegativeValue()
01149         if fc > 0: raise PositiveValue()
01150         if Ec < 0: raise NegativeValue()
01151         if ec != 1 and ec > 0: raise PositiveValue()
01152         if ecp != 1 and ecp > 0: raise PositiveValue()
01153         if fct != -1 and fct < 0: raise NegativeValue()
01154         if et != -1 and et < 0: raise NegativeValue()
01155
01156         # Arguments
01157         self.ID = ID
01158         self.fc = fc
01159         self.Ec = Ec
01160         self.beta = beta
```

```
01161
01162         # Initialized the parameters that are dependent from others
01163         self.section_name_tag = "None"
01164         self.Initialized = False
01165         self.ReInit(ec, ecp, fct, et)
01166
```

7.55.3 Member Function Documentation

7.55.3.1 CheckApplicability()

```
def CheckApplicability (
                self )
```

Implementation of the homonym abstract method.

See parent class MaterialModels for detailed information.

Reimplemented from MaterialModels.

Definition at line 1242 of file MaterialModels.py.

```
01242     def CheckApplicability(self):
01243         """
01244         Implementation of the homonym abstract method.
01245         See parent class MaterialModels for detailed information.
01246         """
01247         Check = True
01248         if self.fc < -110*MPa_unit: # Deierlein 1999
01249             Check = False
01250             print("With High Strength concrete (< -110 MPa), a better material model should be used
       (see Abdesselam et Al. 2019")
01251         if not Check:
01252             print("The validity of the equations is not fullfilled.")
01253             print("!!!!!!! WARNING !!!!!!! Check material model of Unconfined Mander 1988, ID=",
       self.ID)
01254             print("")
01255
01256
```

7.55.3.2 Compute_ec()

```
def Compute_ec (
                self )
```

Method that computes the compressive concrete yield strain.

For more information, see Karthik and Mander 2011.

Returns

float: Strain

Definition at line 1257 of file MaterialModels.py.

```
01257     def Compute_ec(self):
01258         """
01259         Method that computes the compressive concrete yield strain.
01260         For more information, see Karthik and Mander 2011.
01261
01262         @returns float: Strain
01263         """
01264         # return -0.002 # Alternative: Mander et Al. 1988
01265         return -0.0015 + self.fc/MPa_unit/70000 # Karthik Mander 2011
01266
```

7.55.3.3 Compute_ecp()

```
def Compute_ecp (
              self )
```

Method that computes the compressive concrete spalling strain.

For more information, see Mander et Al. 1988.

Returns

float: Strain

Definition at line 1267 of file MaterialModels.py.

```
01267     def Compute_ecp(self):
01268         """
01269         Method that computes the compressive concrete spalling strain.
01270         For more information, see Mander et Al. 1988.
01271 
01272         @returns float: Strain
01273         """
01274         return 2.0*self.ec
01275 
01276 
```

7.55.3.4 Compute_ecu()

```
def Compute_ecu (
              self )
```

Method that computes the compressive concrete failure strain.

For more information, see Karthik and Mander 2011.

Returns

float: Strain

Definition at line 1297 of file MaterialModels.py.

```
01297     def Compute_ecu(self):
01298         """
01299         Method that computes the compressive concrete failure strain.
01300         For more information, see Karthik and Mander 2011.
01301 
01302         @returns float: Strain
01303         """
01304         # return -0.004 # Alternative: Mander et Al. 1988
01305         return -0.012 - 0.0001 * self.fc/MPa_unit # Karthik Mander 2011
01306 
```

7.55.3.5 Compute_et()

```
def Compute_et (
              self )
```

Method that computes the tensile concrete yield strain.

For more information, see Mander et Al. 1988 (eq 45).

Returns

float: Strain.

Definition at line 1287 of file MaterialModels.py.

```
    def Compute_et(self):
        """
        Method that computes the tensile concrete yield strain.
        For more information, see Mander et Al. 1988 (eq 45).

        @returns float: Strain.
        """
        return self.fct/self.Ec


```

7.55.3.6 Compute_fct()

```
def Compute_fct (
              self )
```

Method that computes the tensile concrete yield stress.

For more information, see SIA 262:2012.

Returns

float: Stress.

Definition at line 1277 of file MaterialModels.py.

```
    def Compute_fct(self):
        """
        Method that computes the tensile concrete yield stress.
        For more information, see SIA 262:2012.

        @returns float: Stress.
        """
        return 0.30 * math.pow(-self.fc/MPa_unit, 2/3) * MPa_unit


```

7.55.3.7 Concrete01()

```
def Concrete01 (
              self )
```

Generate the material model Concrete01 for unconfined concrete using the computed parameters.

See _Concrete01 function for more information. Use this method or Concrete04, not both (only one material model for ID).

Definition at line 1307 of file MaterialModels.py.

```
01307     def Concrete01(self):
01308         """
01309         Generate the material model Concrete01 for unconfined concrete using the computed parameters.
01310         See _Concrete01 function for more information. Use this method or Concrete04, not both (only one material model for ID).
01311         """
01312         _Concrete01(self.ID, self.ec, self.fc, self.ecu)
01313         self.Initialized = True
01314         self.UpdateStoredData()
01315 
01316 
```

7.55.3.8 Concrete04()

```
def Concrete04 (
              self )
```

Generate the material model Concrete04 for unconfined concrete (Mander 1988) using the computed parameters.

See _Concrete04 function for more information. Use this method or Concrete01, not both (only one material model for ID).

Definition at line 1317 of file MaterialModels.py.

```
01317     def Concrete04(self):
01318         """
01319         Generate the material model Concrete04 for unconfined concrete (Mander 1988) using the computed parameters.
01320         See _Concrete04 function for more information. Use this method or Concrete01, not both (only one material model for ID).
01321         """
01322         _Concrete04(self.ID, self.fc, self.ec, self.ecu, self.Ec, self.fct, self.et, self.beta)
01323         self.Initialized = True
01324         self.UpdateStoredData()
01325 
01326 
```

7.55.3.9 ReInit()

```
def ReInit (
              self,
              ec = 1,
              ecp = 1,
              fct = -1,
              et = -1 )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

ec	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
ecp	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
fct	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
et	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.

Definition at line 1168 of file MaterialModels.py.

```
01168     def ReInit(self, ec = 1, ecp = 1, fct = -1, et = -1):
01169         """
01170         Implementation of the homonym abstract method.
01171         See parent class DataManagement for detailed information.
01172 
01173         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
01174         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
01175         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
01176         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
01177         """
01178         # Check applicability
01179         self.CheckApplicability()
01180 
01181         # Arguments
01182         self.ec = self.Compute_ec() if ec == 1 else ec
01183         self.ecp = self.Compute_ecp() if ecp == 1 else ecp
01184         self.fct = self.Compute_fct() if fct == -1 else fct
01185         self.et = self.Compute_et() if et == -1 else et
01186 
01187         # Members
01188         self.ecu = self.Compute_ecu()
01189         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + " (modified)"
01190 
01191         # Data storage for loading/saving
01192         self.UpdateStoredData()
01193 
01194 
```

7.55.3.10 ShowInfo()

```
def ShowInfo (
        self,
        plot = False,
        block = False,
        concrete04 = True )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the material model. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
concrete04	(bool, optional): Option to show in the plot the concrete04 or concrete01 if False. Defaults to True.

Definition at line 1214 of file MaterialModels.py.

```
    def ShowInfo(self, plot = False, block = False, concrete04 = True):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        @param concrete04 (bool, optional): Option to show in the plot the concrete04 or concrete01 if False. Defaults to True.
        """
        print("")
        print("Requested info for Unconfined Mander 1988 material model Parameters, ID = {}".format(self.ID))
        print("Section associated: {} ".format(self.section_name_tag))
        print('Concrete strength fc = {} MPa'.format(self.fc/MPa_unit))
        print('Strain at maximal strength ec = {}'.format(self.ec))
        print('Maximal strain ecu = {}'.format(self.ecu))
        print("")

        if plot:
            fig, ax = plt.subplots()
            if concrete04:
                PlotConcrete04(self.fc, self.Ec, self.ec, self.ecu, "U", ax, self.ID)
            else:
                PlotConcrete01(self.fc, self.ec, 0, self.ecu, ax, self.ID)

            if block:
                plt.show()


```

7.55.3.11 UpdateStoredData()

```
def UpdateStoredData (
              self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 1195 of file MaterialModels.py.

```
01195     def UpdateStoredData(self):
01196         """
01197         Implementation of the homonym abstract method.
01198         See parent class DataManagement for detailed information.
01199         """
01200         self.data = [["INFO_TYPE", "UnconfMander1988"], # Tag for differentiating different data
01201             ["ID", self.ID],
01202             ["section_name_tag", self.section_name_tag],
01203             ["fc", self.fc],
01204             ["Ec", self.Ec],
01205             ["ec", self.ec],
01206             ["ecp", self.ecp],
01207             ["ecu", self.ecu],
01208             ["fct", self.fct],
01209             ["et", self.et],
01210             ["beta", self.beta],
01211             ["Initialized", self.Initialized]]
01212
01213
```

7.55.4 Member Data Documentation

7.55.4.1 beta

```
beta
```

Definition at line 1160 of file MaterialModels.py.

7.55.4.2 data

```
data
```

Definition at line 1200 of file MaterialModels.py.

7.55.4.3 Ec

```
Ec
```

Definition at line 1159 of file MaterialModels.py.

7.55.4.4 ec

```
ec
```

Definition at line 1182 of file MaterialModels.py.

7.55.4.5 ecp

```
ecp
```

Definition at line 1183 of file MaterialModels.py.

7.55.4.6 ecu

```
ecu
```

Definition at line 1188 of file MaterialModels.py.

7.55.4.7 et

```
et
```

Definition at line 1185 of file MaterialModels.py.

7.55.4.8 fc

```
fc
```

Definition at line 1158 of file MaterialModels.py.

7.55.4.9 fct

```
fct
```

Definition at line 1184 of file MaterialModels.py.

7.55.4.10 ID

```
ID
```

Definition at line 1157 of file MaterialModels.py.

7.55.4.11 Initialized

```
Initialized
```

Definition at line 1164 of file MaterialModels.py.

7.55.4.12 section_name_tag

```
section_name_tag
```

Definition at line 1163 of file MaterialModels.py.

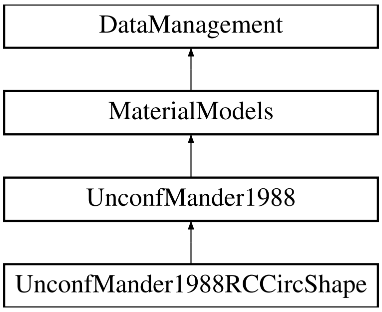
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.56 UnconfMander1988RCCircShape Class Reference

Class that is the children of UnconfMander1988 and combine the class RCCircShape (section) to retrieve the information needed.

Inheritance diagram for UnconfMander1988RCCircShape:



Public Member Functions

- def __init__ (self, int ID, RCCircShape section, ec=1, ecp=1, fct=-1, et=-1, beta=0.1)
  Constructor of the class.

Public Attributes

- section
- section_name_tag

7.56.1 Detailed Description

Class that is the children of UnconfMander1988 and combine the class RCCircShape (section) to retrieve the information needed.

Parameters

UnconfMander1988	Parent class.

Definition at line 1354 of file MaterialModels.py.

7.56.2 Constructor & Destructor Documentation

7.56.2.1 __init__()

```
def __init__ (
              self,
             int ID,
             RCCircShape section,
              ec = 1,
              ecp = 1,
              fct = -1,
              et = -1,
              beta = 0.1 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class RCCircShape. The copy of the section passed is stored in the member variable self.section.

Parameters

ID	(int): Unique material model ID.
section	(RCCircShape): RCCircShape section object.
ec	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
ecp	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
fct	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
et	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
beta	(float, optional): Loating point value defining the exponential curve parameter to define the residual stress. Defaults to 0.1 (according to OpenSeesPy documentation)

Reimplemented from UnconfMander1988.

Definition at line 1360 of file MaterialModels.py.

```
    def __init__(self, ID: int, section: RCCircShape, ec=1, ecp=1, fct=-1, et=-1, beta=0.1):
        """
        Constructor of the class. It passes the arguments into the parent class to generate the
combination of the parent class
            and the section class RCCircShape.
        The copy of the section passed is stored in the member variable self.section.

        @param ID (int): Unique material model ID.
        @param section (RCCircShape): RCCircShape section object.
        @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
according to Karthik and Mander 2011.
        @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
to Mander 1988.
        @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
according to SIA 262:2012.
        @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
according to SIA 262:2012.
        @param beta (float, optional): Loating point value defining the exponential curve parameter to
define the residual stress.
            Defaults to 0.1 (according to OpenSeesPy documentation)
        """
        self.section = deepcopy(section)
        super().__init__(ID, section.fc, section.Ec, ec=ec, ecp=ecp, fct=fct, et=et, beta=beta)
        self.section_name_tag = section.name_tag
        self.UpdateStoredData()
```

7.56.3 Member Data Documentation

7.56.3.1 section

`section`

Definition at line 1375 of file MaterialModels.py.

7.56.3.2 section_name_tag

`section_name_tag`

Definition at line 1377 of file MaterialModels.py.

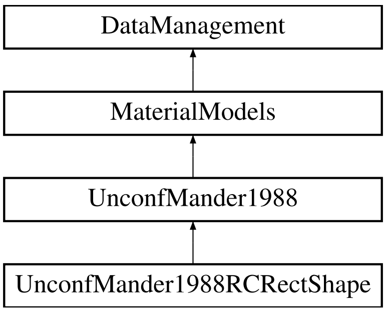
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.57 UnconfMander1988RCRectShape Class Reference

Class that is the children of UnconfMander1988 and combine the class RCRectShape (section) to retrieve the information needed.

Inheritance diagram for UnconfMander1988RCRectShape:



Public Member Functions

- def __init__ (self, int ID, RCRectShape section, ec=1, ecp=1, fct=-1, et=-1, beta=0.1)

  Constructor of the class.

Public Attributes

- section
- section_name_tag

7.57.1 Detailed Description

Class that is the children of UnconfMander1988 and combine the class RCRectShape (section) to retrieve the information needed.

Parameters

UnconfMander1988	Parent class.

Definition at line 1327 of file MaterialModels.py.

7.57.2 Constructor & Destructor Documentation

7.57.2.1 __init__()

```
def __init__ (
        self,
        int ID,
        RCRectShape section,
        ec = 1,
        ecp = 1,
        fct = -1,
        et = -1,
        beta = 0.1 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class RCRectShape. The copy of the section passed is stored in the member variable self.section.

Parameters

ID	(int): Unique material model ID.
section	(RCRectShape): RCRectShape section object.
ec	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
ecp	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
fct	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
et	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
beta	(float, optional): Loating point value defining the exponential curve parameter to define the residual stress. Defaults to 0.1 (according to OpenSeesPy documentation)

Reimplemented from UnconfMander1988.

Definition at line 1333 of file MaterialModels.py.

```
01333     def __init__(self, ID: int, section: RCRectShape, ec=1, ecp=1, fct=-1, et=-1, beta=0.1):
01334         """
01335         Constructor of the class. It passes the arguments into the parent class to generate the
       combination of the parent class
01336             and the section class RCRectShape.
01337         The copy of the section passed is stored in the member variable self.section.
01338 
01339         @param ID (int): Unique material model ID.
01340         @param section (RCRectShape): RCRectShape section object.
```

```
01341         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
     according to Karthik and Mander 2011.
01342         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
     to Mander 1988.
01343         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
     according to SIA 262:2012.
01344         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
     according to SIA 262:2012.
01345         @param beta (float, optional): Loating point value defining the exponential curve parameter to
     define the residual stress.
01346             Defaults to 0.1 (according to OpenSeesPy documentation)
01347         """
01348         self.section = deepcopy(section)
01349         super().__init__(ID, section.fc, section.Ec, ec=ec, ecp=ecp, fct=fct, et=et, beta=beta)
01350         self.section_name_tag = section.name_tag
01351         self.UpdateStoredData()
01352
01353
```

7.57.3 Member Data Documentation

7.57.3.1 section

`section`

Definition at line 1348 of file MaterialModels.py.

7.57.3.2 section_name_tag

`section_name_tag`

Definition at line 1350 of file MaterialModels.py.

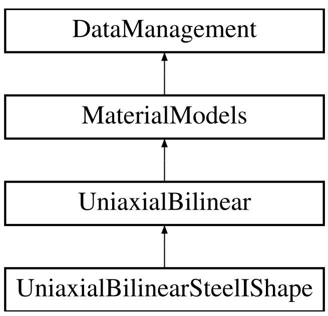
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.58 UniaxialBilinear Class Reference

Class that stores funcions and material properties of a simple uniaxial bilinear model with the OpenSeesPy command type used to model it is Steel01.

Inheritance diagram for UniaxialBilinear:



Public Member Functions

- def __init__ (self, int ID, fy, Ey, b=0.01)

    Constructor of the class.
- def CheckApplicability (self)

    Implementation of the homonym abstract method.
- def ReInit (self)

    Implementation of the homonym abstract method.
- def ShowInfo (self, plot=False, block=False)

    Implementation of the homonym abstract method.
- def Steel01 (self)

    Generate the material model Steel01 uniaxial bilinear material model.
- def UpdateStoredData (self)

    Implementation of the homonym abstract method.

Public Attributes

- b
- data
- Ey
- ey
- fy
- ID
- Initialized
- section_name_tag

7.58.1 Detailed Description

Class that stores funcions and material properties of a simple uniaxial bilinear model with the OpenSeesPy command type used to model it is Steel01.

Parameters

MaterialModels	Parent abstract class.

Definition at line 2319 of file MaterialModels.py.

7.58.2 Constructor & Destructor Documentation

7.58.2.1 __init__()

```
def __init__ (
              self,
             int ID,
              fy,
```

```
Ey,
b = 0.01 )
```

Constructor of the class.

Parameters

ID	(int): Unique material model ID.
fy	(float): Yield stress.
Ey	(float): Young modulus.
b	(float, optional): Strain hardening factor. Defaults to 0.01.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	fy needs to be positive.
NegativeValue	Ey needs to be positive.

Reimplemented in UniaxialBilinearSteelIShape.

Definition at line 2326 of file MaterialModels.py.

```
02326     def __init__(self, ID: int, fy, Ey, b = 0.01):
02327         """
02328         Constructor of the class.
02329 
02330         @param ID (int): Unique material model ID.
02331         @param fy (float): Yield stress.
02332         @param Ey (float): Young modulus.
02333         @param b (float, optional): Strain hardening factor. Defaults to 0.01.
02334 
02335         @exception NegativeValue: ID needs to be a positive integer.
02336         @exception NegativeValue: fy needs to be positive.
02337         @exception NegativeValue: Ey needs to be positive.
02338         """
02339         # Check
02340         if ID < 1: raise NegativeValue()
02341         if fy < 0: raise NegativeValue()
02342         if Ey < 0: raise NegativeValue()
02343 
02344         # Arguments
02345         self.ID = ID
02346         self.fy = fy
02347         self.Ey = Ey
02348         self.b = b
02349 
02350         # Initialized the parameters that are dependent from others
02351         self.section_name_tag = "None"
02352         self.Initialized = False
02353         self.ReInit()
02354 
```

7.58.3 Member Function Documentation

7.58.3.1 CheckApplicability()

```
def CheckApplicability (
              self )
```

Implementation of the homonym abstract method.

See parent class MaterialModels for detailed information.

Reimplemented from MaterialModels.

Definition at line 2423 of file MaterialModels.py.

```
    def CheckApplicability(self):
        """
        Implementation of the homonym abstract method.
        See parent class MaterialModels for detailed information.
        """
        Check = True
        # if len(self.wy) == 0 or len(self.wx_top) == 0 or len(self.wx_bottom) == 0:
        #     Check = False
        #     print("Hypothesis of one bar per corner not fullfilled.")
        if not Check:
            print("The validity of the equations is not fullfilled.")
            print("!!!!!!! WARNING !!!!!!! Check material model of Uniaxial Bilinear, ID=", self.ID)
            print("")


```

7.58.3.2 ReInit()

def ReInit (
self)

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 2355 of file MaterialModels.py.

```
    def ReInit(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        # Check applicability
        self.CheckApplicability()

        # Members
        self.ey = self.fy / self.Ey
        if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredData()


```

7.58.3.3 ShowInfo()

def ShowInfo (
self,
plot = False,
block = False)

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Parameters

plot	(bool, optional): Option to show the plot of the material model. Defaults to False.
block	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 2387 of file MaterialModels.py.

```
02387     def ShowInfo(self, plot = False, block = False):
02388         """
02389         Implementation of the homonym abstract method.
02390         See parent class DataManagement for detailed information.
02391 
02392         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
      False.
02393         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
      of the program everytime that a plot should pop up). Defaults to False.
02394         """
02395         print("")
02396         print("Requested info for Uniaxial Bilinear material model Parameters, ID =
      {}".format(self.ID))
02397         print("Section associated: {} ".format(self.section_name_tag))
02398         print('Yielding stress fy = {} MPa'.format(self.fy/MPa_unit))
02399         print('Young modulus Ey = {} MPa'.format(self.Ey/MPa_unit))
02400         print('Maximal elastic strain epsilon y = {}'.format(self.ey))
02401         print('Hardening factor b = {}'.format(self.b))
02402         print("")
02403 
02404         if plot:
02405             # Data for plotting
02406             e_pl = 10.0 * self.ey # to show that if continues with this slope
02407             sigma_pl = self.b * self.Ey * e_pl
02408 
02409             x_axis = np.array([0.0, self.ey, (self.ey+e_pl)])*100
02410             y_axis = np.array([0.0, self.fy, (self.fy+sigma_pl)])/MPa_unit
02411 
02412             fig, ax = plt.subplots()
02413             ax.plot(x_axis, y_axis, 'k-')
02414 
02415             ax.set(xlabel='Strain [%]', ylabel='Stress [MPa]',
02416                 title='Uniaxial Bilinear model for material ID={}'.format(self.ID))
02417             ax.grid()
02418 
02419             if block:
02420                 plt.show()
02421 
02422 
```

7.58.3.4 Steel01()

```
def Steel01 (
             self )
```

Generate the material model Steel01 uniaxial bilinear material model.

See _Steel01 function for more information.

Definition at line 2438 of file MaterialModels.py.

```
02438     def Steel01(self):
02439         """
02440         Generate the material model Steel01 uniaxial bilinear material model.
02441         See _Steel01 function for more information.
02442         """
02443         _Steel01(self.ID, self.fy, self.Ey, self.b)
02444         self.Initialized = True
02445         self.UpdateStoredData()
02446 
02447 
```

7.58.3.5 UpdateStoredData()

```
def UpdateStoredData (
                self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 2372 of file MaterialModels.py.

```
    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.data = [["INFO_TYPE", "UniaxialBilinear"], # Tag for differentiating different data
            ["ID", self.ID],
            ["section_name_tag", self.section_name_tag],
            ["fy", self.fy],
            ["Ey", self.Ey],
            ["ey", self.ey],
            ["b", self.b],
            ["Initialized", self.Initialized]]


```

7.58.4 Member Data Documentation

7.58.4.1 b

```
b
```

Definition at line 2348 of file MaterialModels.py.

7.58.4.2 data

```
data
```

Definition at line 2377 of file MaterialModels.py.

7.58.4.3 Ey

```
Ey
```

Definition at line 2347 of file MaterialModels.py.

7.58.4.4 ey

```
ey
```

Definition at line 2364 of file MaterialModels.py.

7.58.4.5 fy

```
fy
```

Definition at line 2346 of file MaterialModels.py.

7.58.4.6 ID

```
ID
```

Definition at line 2345 of file MaterialModels.py.

7.58.4.7 Initialized

```
Initialized
```

Definition at line 2352 of file MaterialModels.py.

7.58.4.8 section_name_tag

```
section_name_tag
```

Definition at line 2351 of file MaterialModels.py.

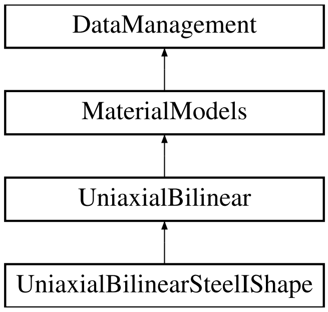
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.59 UniaxialBilinearSteelIShape Class Reference

Class that is the children of UniaxialBilinear and combine the class SteelIShape (section) to retrieve the information needed.

Inheritance diagram for UniaxialBilinearSteelIShape:



Public Member Functions

- def __init__ (self, int ID, SteelIShape section, b=0.01)

    Constructor of the class.

Public Attributes

- section
- section_name_tag

7.59.1 Detailed Description

Class that is the children of UniaxialBilinear and combine the class SteelIShape (section) to retrieve the information needed.

Parameters

UniaxialBilinear	Parent class.

Definition at line 2448 of file MaterialModels.py.

7.59.2 Constructor & Destructor Documentation

7.59.2.1 __init__()

```
def __init__ (
                self,
               int ID,
               SteelIShape section,
                b = 0.01 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class SteelIShape. The copy of the section passed is stored in the member variable self.section.

Parameters

ID	(int): Unique material model ID.
section	(SteelIShape): SteelIShape section object.
b	(float, optional): Strain hardening factor. Defaults to 0.01.

Reimplemented from UniaxialBilinear.

Definition at line 2454 of file MaterialModels.py.

```
    def __init__(self, ID: int, section: SteelIShape, b=0.01):
        """
        Constructor of the class. It passes the arguments into the parent class to generate the
combination of the parent class
            and the section class SteelIShape.
        The copy of the section passed is stored in the member variable self.section.

        @param ID (int): Unique material model ID.
        @param section (SteelIShape): SteelIShape section object.
        @param b (float, optional): Strain hardening factor. Defaults to 0.01.
        """
        self.section = deepcopy(section)
        super().__init__(ID, section.Fy, section.E, b=b)
        self.section_name_tag = section.name_tag
        self.UpdateStoredData()


```

7.59.3 Member Data Documentation

7.59.3.1 section

```
section
```

Definition at line 2464 of file MaterialModels.py.

7.59.3.2 section_name_tag

```
section_name_tag
```

Definition at line 2466 of file MaterialModels.py.

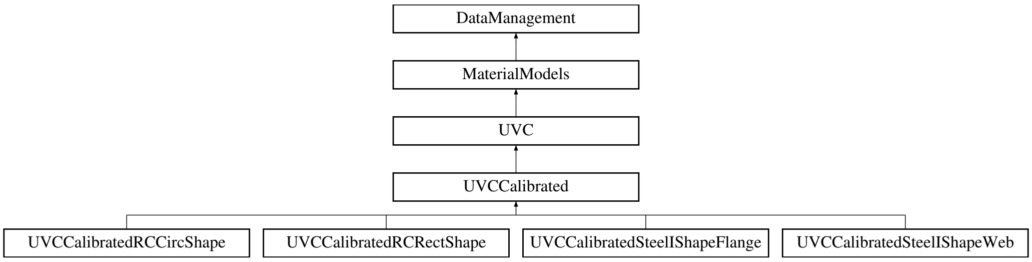
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.60 UVC Class Reference

Class that stores funcions and material properties of a steel profile or reinforcing bar with Updated Voce-Chaboche as the material model and the OpenSeesPy command type used to model it is UVCuniaxial.

Inheritance diagram for UVC:



Public Member Functions

- def __init__ (self, int ID, fy, Ey, QInf, b, DInf, a, np.ndarray cK, np.ndarray gammaK)

    Constructor of the class.
- def CheckApplicability (self)

    Implementation of the homonym abstract method.
- def ReInit (self)

    Implementation of the homonym abstract method.
- def ShowInfo (self)

    Implementation of the homonym abstract method.
- def UpdateStoredData (self)

    Implementation of the homonym abstract method.
- def UVCuniaxial (self)

    Generate the material model Updated Voce-Chaboche (UVC) for uniaxial stress states.

Public Attributes

- a
- b
- cK
- data
- DInf
- Ey
- fy
- gammaK
- ID
- Initialized
- N
- QInf
- section_name_tag

7.60.1 Detailed Description

Class that stores funcions and material properties of a steel profile or reinforcing bar with Updated Voce-Chaboche as the material model and the OpenSeesPy command type used to model it is UVCuniaxial.

For more information about the how to calibrate the set of parameters, see de Castro e Sousa, Suzuki and Lignos 2020 and Hartloper, de Castro e Sousa and Lignos 2021.

Parameters

MaterialModels	Parent abstract class.

Definition at line 2629 of file MaterialModels.py.

7.60.2 Constructor & Destructor Documentation

7.60.2.1 __init__()

```
def __init__ (
             self,
            int ID,
             fy,
             Ey,
             QInf,
             b,
             DInf,
             a,
            np.ndarray cK,
            np.ndarray gammaK )
```

Constructor of the class.

Parameters

ID	(int): Unique material model ID.
fy	(float): Initial yield stress of the steel material.
Ey	(float): Elastic modulus of the steel material.
QInf	(float): Maximum increase in yield stress due to cyclic hardening (isotropic hardening).
b	(float): Saturation rate of QInf.
DInf	(float): Decrease in the initial yield stress, to neglect the model updates set DInf = 0.
a	(float): Saturation rate of DInf, a $>$ 0. If DInf == 0, then a is arbitrary (but still a $>$ 0).
cK	(np.ndarray): Array of 1 dimension; each entry is one kinematic hardening parameter associated with one backstress, up to 8 may be specified.
gammaK	(np.ndarray): Array of 1 dimension; each entry is one saturation rate of kinematic hardening associated with one backstress, up to 8 may be specified.

Exceptions

NegativeValue	ID needs to be a positive integer.
NegativeValue	fy needs to be positive.
NegativeValue	Ey needs to be positive.
NegativeValue	QInf needs to be positive.
NegativeValue	b needs to be positive.
NegativeValue	DInf needs to be positive.
NegativeValue	a needs to be positive.

Exceptions

WrongArgument	cK can't be empty.
WrongArgument	cK and gammaK have as many entries as the number of backstresses (thus they have the same length).

Reimplemented in UVCCalibratedRCCircShape, UVCCalibratedRCRectShape, UVCCalibratedSteelIShapeFlange, UVCCalibratedSteelIShapeWeb, and UVCCalibrated.

Definition at line 2638 of file MaterialModels.py.

```
02638     def __init__(self, ID: int, fy, Ey, QInf, b, DInf, a, cK: np.ndarray, gammaK: np.ndarray):
02639         """
02640         Constructor of the class.
02641 
02642         @param ID (int): Unique material model ID.
02643         @param fy (float): Initial yield stress of the steel material.
02644         @param Ey (float): Elastic modulus of the steel material.
02645         @param QInf (float): Maximum increase in yield stress due to cyclic hardening (isotropic
hardening).
02646         @param b (float): Saturation rate of QInf.
02647         @param DInf (float): Decrease in the initial yield stress, to neglect the model updates set
DInf = 0.
02648         @param a (float): Saturation rate of DInf, a > 0. If DInf == 0, then a is arbitrary (but still
a > 0).
02649         @param cK (np.ndarray): Array of 1 dimension; each entry is one kinematic hardening parameter
associated with one backstress, up to 8 may be specified.
02650         @param gammaK (np.ndarray): Array of 1 dimension; each entry is one saturation rate of
kinematic hardening associated with one backstress, up to 8 may be specified.
02651 
02652         @exception NegativeValue: ID needs to be a positive integer.
02653         @exception NegativeValue: fy needs to be positive.
02654         @exception NegativeValue: Ey needs to be positive.
02655         @exception NegativeValue: QInf needs to be positive.
02656         @exception NegativeValue: b needs to be positive.
02657         @exception NegativeValue: DInf needs to be positive.
02658         @exception NegativeValue: a needs to be positive.
02659         @exception WrongArgument: cK can't be empty.
02660         @exception WrongArgument: cK and gammaK have as many entries as the number of backstresses
(thus they have the same length).
02661         """
02662         # Check
02663         if ID < 1: raise NegativeValue()
02664         if fy < 0: raise NegativeValue()
02665         if Ey < 0: raise NegativeValue()
02666         if QInf < 0: raise NegativeValue()
02667         if b < 0: raise NegativeValue()
02668         if DInf < 0: raise NegativeValue()
02669         if a < 0: raise NegativeValue()
02670         if len(cK) == 0: raise WrongArgument()
02671         if len(cK) != len(gammaK): raise WrongArgument()
02672         if len(cK) != 2: print("!!!!!!! WARNING !!!!!!! Number of backstresses should be 2 for optimal
performances")
02673         if DInf == 0: print("!!!!!!! WARNING !!!!!!! With DInf = 0, the model used is Voce-Chaboche
(VC) not updated (UVC)")
02674 
02675         # Arguments
02676         self.ID = ID
02677         self.fy = fy
02678         self.Ey = Ey
02679         self.QInf = QInf
02680         self.b = b
02681         self.DInf = DInf
02682         self.a = a
02683         self.cK = copy(cK)
02684         self.gammaK = copy(gammaK)
02685 
02686         # Initialized the parameters that are dependent from others
02687         self.section_name_tag = "None"
02688         self.Initialized = False
02689         self.ReInit()
02690 
```

7.60.3 Member Function Documentation

7.60.3.1 CheckApplicability()

```
def CheckApplicability (
                self )
```

Implementation of the homonym abstract method.

See parent class MaterialModels for detailed information.

Reimplemented from MaterialModels.

Definition at line 2747 of file MaterialModels.py.

```
02747     def CheckApplicability(self):
02748         """
02749         Implementation of the homonym abstract method.
02750         See parent class MaterialModels for detailed information.
02751         """
02752         Check = True
02753         # No checks
02754         if not Check:
02755             print("The validity of the equations is not fullfilled.")
02756             print("!!!!!!! WARNING !!!!!!! Check material model of UVC, ID=", self.ID)
02757             print("")
02758 
02759 
```

7.60.3.2 ReInit()

```
def ReInit (
                self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 2691 of file MaterialModels.py.

```
02691     def ReInit(self):
02692         """
02693         Implementation of the homonym abstract method.
02694         See parent class DataManagement for detailed information.
02695         """
02696         # Check applicability
02697         self.CheckApplicability()
02698 
02699         # Members
02700         self.N = len(self.cK)
02701         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
       (modified)"
02702 
02703         # Data storage for loading/saving
02704         self.UpdateStoredData()
02705 
02706 
```

7.60.3.3 ShowInfo()

```
def ShowInfo (
            self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 2728 of file MaterialModels.py.

```
02728     def ShowInfo(self):
02729         """
02730         Implementation of the homonym abstract method.
02731         See parent class DataManagement for detailed information.
02732         """
02733         print("")
02734         print("Requested info for UVC material model Parameters, ID = {}".format(self.ID))
02735         print("Section associated: {} ".format(self.section_name_tag))
02736         print("Yield strength fy = {} MPa".format(self.fy/MPa_unit))
02737         print("Young modulus Ey = {} MPa".format(self.Ey/MPa_unit))
02738         print("Isotropic hardening factor QInf = {} MPa and saturation rate  b =
      {}".format(self.QInf/MPa_unit, self.b))
02739         print("Decrease the initial yield stress DInf = {} MPa and saturation rate a =
      {}".format(self.DInf/MPa_unit, self.a))
02740         print("Kinematic hardening vector ({} backstresses) cK = {} MPa".format(self.N,
      self.cK/MPa_unit))
02741         print("And associated saturation rate gammaK = {}".format(self.gammaK))
02742         print("")
02743
02744         #TODO: implement plot (too complex for now)
02745
02746
```

7.60.3.4 UpdateStoredData()

```
def UpdateStoredData (
            self )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

Definition at line 2708 of file MaterialModels.py.

```
02708     def UpdateStoredData(self):
02709         """
02710         Implementation of the homonym abstract method.
02711         See parent class DataManagement for detailed information.
02712         """
02713         self.data = [["INFO_TYPE", "UVC"], # Tag for differentiating different data
02714             ["ID", self.ID],
02715             ["section_name_tag", self.section_name_tag],
02716             ["fy", self.fy],
02717             ["Ey", self.Ey],
02718             ["QInf", self.QInf],
02719             ["b", self.b],
02720             ["DInf", self.DInf],
02721             ["a", self.a],
02722             ["N", self.N],
02723             ["ck", self.cK],
02724             ["gammaK", self.gammaK],
02725             ["Initialized", self.Initialized]]
02726
02727
```

7.60.3.5 UVCuniaxial()

```
def UVCuniaxial (
              self )
```

Generate the material model Updated Voce-Chaboche (UVC) for uniaxial stress states.

See _UVCuniaxial function for more information.

Definition at line 2760 of file MaterialModels.py.

```
02760     def UVCuniaxial(self):
02761         """
02762         Generate the material model Updated Voce-Chaboche (UVC) for uniaxial stress states.
02763         See _UVCuniaxial function for more information.
02764         """
02765         _UVCuniaxial(self.ID, self.Ey, self.fy, self.QInf, self.b, self.DInf, self.a, self.N, self.cK,
       self.gammaK)
02766         self.Initialized = True
02767         self.UpdateStoredData()
02768
02769
```

7.60.4 Member Data Documentation

7.60.4.1 a

```
a
```

Definition at line 2682 of file MaterialModels.py.

7.60.4.2 b

```
b
```

Definition at line 2680 of file MaterialModels.py.

7.60.4.3 cK

```
cK
```

Definition at line 2683 of file MaterialModels.py.

7.60.4.4 data

```
data
```

Definition at line 2713 of file MaterialModels.py.

7.60.4.5 DInf

```
DInf
```

Definition at line 2681 of file MaterialModels.py.

7.60.4.6 Ey

```
Ey
```

Definition at line 2678 of file MaterialModels.py.

7.60.4.7 fy

```
fy
```

Definition at line 2677 of file MaterialModels.py.

7.60.4.8 gammaK

```
gammaK
```

Definition at line 2684 of file MaterialModels.py.

7.60.4.9 ID

```
ID
```

Definition at line 2676 of file MaterialModels.py.

7.60.4.10 Initialized

```
Initialized
```

Definition at line 2688 of file MaterialModels.py.

7.60.4.11 N

```
N
```

Definition at line 2700 of file MaterialModels.py.

7.60.4.12 QInf

```
QInf
```

Definition at line 2679 of file MaterialModels.py.

7.60.4.13 section_name_tag

```
section_name_tag
```

Definition at line 2687 of file MaterialModels.py.

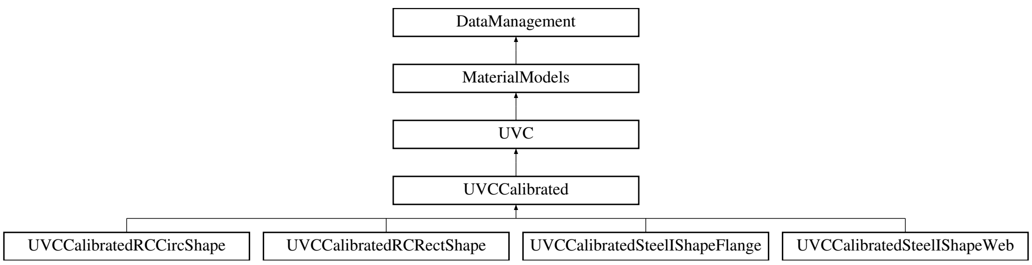
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.61 UVCCalibrated Class Reference

Class that is the children of UVC that retrieve calibrated parameters from UVC_calibrated_parameters.txt.

Inheritance diagram for UVCCalibrated:



Public Member Functions

- def __init__ (self, int ID, str calibration, fy=-1, E=-1)

  Constructor of the class.

Public Attributes

- calibration

7.61.1 Detailed Description

Class that is the children of UVC that retrieve calibrated parameters from UVC_calibrated_parameters.txt.

The file text can be modified by adding more calibrated parameters.

Parameters

UVC	Parent class.

Definition at line 2770 of file MaterialModels.py.

7.61.2 Constructor & Destructor Documentation

7.61.2.1 __init__()

```
def __init__ (
             self,
            int ID,
            str calibration,
             fy = -1,
             E = -1 )
```

Constructor of the class.

It retrieve the parameters from UVC_calibrated_parameters.txt and pass them in the parent class.

Parameters

ID	(int): Unique material model ID.
calibration	(str): Label of the calibration parameter set. The options are:

7.61.3 'S355J2_25mm_plate' \n

7.61.4 'S355J2_50mm_plate' \n

7.61.5 'S355J2_HEB500_flange' \n

7.61.6 'S355J2_HEB500_web' \n

7.61.7 'S460NL_25mm_plate' \n

7.61.8 'S690QL_25mm_plate' \n

7.61.9 'A992Gr50_W14X82_web' \n

7.61.10 'A992Gr50_W14X82_flange' \n

7.61.11 'A500GrB_HSS305X16' \n

7.61.12 'BCP325_22mm_plate' \n

7.61.13 'BCR295_HSS350X22' \n

7.61.14 'HYP400_27mm_plate' \n

Parameters

fy	(float, optional): Yield strength. Defaults to -1, e.g. taken equal to the one given in the calibration parameter set.
E	(float, optional): Young modulus. Defaults to -1, e.g. taken equal to the one given in the calibration parameter set.

Exceptions

NegativeValue	fy needs to be positive if different from -1.
NegativeValue	E needs to be positive if different from -1.
NameError	calibration needs to be equal to the label of one of the set of calibrated parameters.

Reimplemented from UVC.

Reimplemented in UVCCalibratedRCCircShape, UVCCalibratedRCRectShape, UVCCalibratedSteelIShapeFlange, and UVCCalibratedSteelIShapeWeb.

Definition at line 2777 of file MaterialModels.py.

```
02777     def __init__(self, ID: int, calibration: str, fy = -1, E = -1):
02778         """
02779         Constructor of the class. It retrieve the parameters from UVC_calibrated_parameters.txt and
      pass them in the parent class.
```

```
02780
02781        @param ID (int): Unique material model ID.
02782        @param calibration (str): Label of the calibration parameter set. The options are: \n
02783        # 'S355J2_25mm_plate' \n
02784        # 'S355J2_50mm_plate' \n
02785        # 'S355J2_HEB500_flange' \n
02786        # 'S355J2_HEB500_web' \n
02787        # 'S460NL_25mm_plate' \n
02788        # 'S690QL_25mm_plate' \n
02789        # 'A992Gr50_W14X82_web' \n
02790        # 'A992Gr50_W14X82_flange' \n
02791        # 'A500GrB_HSS305X16' \n
02792        # 'BCP325_22mm_plate' \n
02793        # 'BCR295_HSS350X22' \n
02794        # 'HYP400_27mm_plate' \n
02795        @param fy (float, optional): Yield strength. Defaults to -1, e.g. taken equal to the one given
    in the calibration parameter set.
02796        @param E (float, optional): Young modulus. Defaults to -1, e.g. taken equal to the one given
    in the calibration parameter set.
02797
02798        @exception NegativeValue: fy needs to be positive if different from -1.
02799        @exception NegativeValue: E needs to be positive if different from -1.
02800        @exception NameError: calibration needs to be equal to the label of one of the set of
    calibrated parameters.
02801        """
02802        if fy != -1 and fy < 0: raise NegativeValue()
02803        if E != -1 and E < 0: raise NegativeValue()
02804
02805        self.calibration = calibration
02806
02807        # Structure of the data to be stored
02808        names = ["Material", "Ey", "fy", "QInf", "b","DInf", "a", "C1", "gamma1", "C2", "gamma2"]
02809        # Get the data
02810        __location__ = os.path.realpath(os.path.join(os.getcwd(), os.path.dirname(__file__)))
02811        UVC_data = np.genfromtxt(os.path.join(__location__, 'UVC_calibrated_parameters.txt'),
    dtype=None, skip_header=1, names = names, encoding='ascii', delimiter='\t')
02812        # Define the index (with the location of the correct set of parameters)
02813        index = UVC_data["Material"] == calibration
02814        fy = UVC_data["fy"][index][0]*MPa_unit if fy == -1 else fy
02815        E = UVC_data["Ey"][index][0]*GPa_unit if E == -1 else E
02816        # Check
02817        if not index.any(): raise NameError("No calibrated parameters with that name. Note that there
    are no spaces in the label.")
02818
02819        # Assign arguments value
02820        super().__init__(ID, fy, E, UVC_data["QInf"][index][0]*MPa_unit, UVC_data["b"][index][0],
02821            UVC_data["DInf"][index][0]*MPa_unit, UVC_data["a"][index][0],
02822            np.array([UVC_data["C1"][index][0], UVC_data["C2"][index][0]])*MPa_unit,
02823            np.array([UVC_data["gamma1"][index][0], UVC_data["gamma2"][index][0]]))
02824
02825
```

7.61.15 Member Data Documentation

7.61.15.1 calibration

```
calibration
```

Definition at line 2805 of file MaterialModels.py.

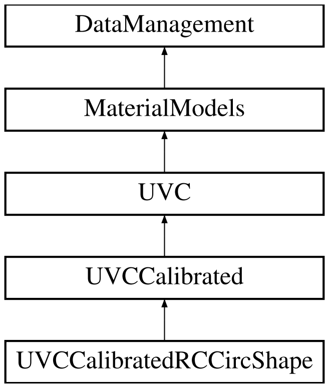
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.62 UVCCalibratedRCCircShape Class Reference

Class that is the children of UVCCalibrated and combine the class RCCircShape (section) to retrieve the information needed.

Inheritance diagram for UVCCalibratedRCCircShape:



Public Member Functions

- def __init__ (self, int ID, RCCircShape section, calibration='S460NL_25mm_plate')

  Constructor of the class.

Public Attributes

- section
- section_name_tag

7.62.1 Detailed Description

Class that is the children of UVCCalibrated and combine the class RCCircShape (section) to retrieve the information needed.

Parameters

UVCCalibrated	Parent class.

Definition at line 2847 of file MaterialModels.py.

7.62.2 Constructor & Destructor Documentation

7.62.2.1 __init__()

```
def __init__ (
              self,
              int ID,
              RCCircShape section,
              calibration = 'S460NL_25mm_plate' )
```

Constructor of the class.

Parameters

ID	(int): Unique material model ID.
section	(RCCircShape): RCCircShape section object.
calibration	(str, optional): Label of the calibration parameter set. The options are listed in UVCCalibrated. Defaults to 'S460NL_25mm_plate'. Change it accordingly to the steel rebars material properties.

Reimplemented from UVCCalibrated.

Definition at line 2853 of file MaterialModels.py.

```
02853     def __init__(self, ID: int, section: RCCircShape, calibration = 'S460NL_25mm_plate'):
02854         """
02855         Constructor of the class.
02856 
02857         @param ID (int): Unique material model ID.
02858         @param section (RCCircShape): RCCircShape section object.
02859         @param calibration (str, optional): Label of the calibration parameter set. The options are
       listed in UVCCalibrated.
02860             Defaults to 'S460NL_25mm_plate'. Change it accordingly to the steel rebars material
       properties.
02861         """
02862         self.section = deepcopy(section)
02863         super().__init__(ID, calibration, section.fy, section.Ey)
02864         self.section_name_tag = section.name_tag
02865         self.UpdateStoredData()
02866 
02867 
```

7.62.3 Member Data Documentation

7.62.3.1 section

```
section
```

Definition at line 2862 of file MaterialModels.py.

7.62.3.2 section_name_tag

```
section_name_tag
```

Definition at line 2864 of file MaterialModels.py.

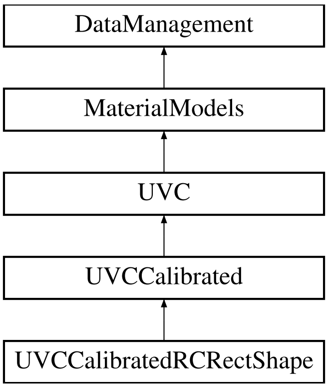
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.63 UVCCalibratedRCRectShape Class Reference

Class that is the children of UVCCalibrated and combines the class RCRectShape (section) to retrieve the information needed.

Inheritance diagram for UVCCalibratedRCRectShape:



Public Member Functions

- def __init__ (self, int ID, RCRectShape section, calibration='S460NL_25mm_plate')

    Constructor of the class.

Public Attributes

- section
- section_name_tag

7.63.1 Detailed Description

Class that is the children of UVCCalibrated and combines the class RCRectShape (section) to retrieve the information needed.

Parameters

UVCCalibrated	Parent class.

Definition at line 2826 of file MaterialModels.py.

7.63.2 Constructor & Destructor Documentation

7.63.2.1 __init__()

```
def __init__ (
              self,
             int ID,
             RCRectShape section,
              calibration = 'S460NL_25mm_plate' )
```

Constructor of the class.

Parameters

ID	(int): Unique material model ID.
section	(RCRectShape): RCRectShape section object.
calibration	(str): Label of the calibration parameter set. The options are listed in UVCCalibrated. Defaults to 'S460NL_25mm_plate'. Change it accordingly to the steel rebars material properties.

Reimplemented from UVCCalibrated.

Definition at line 2832 of file MaterialModels.py.

```
02832     def __init__(self, ID: int, section: RCRectShape, calibration = 'S460NL_25mm_plate'):
02833         """
02834         Constructor of the class.
02835
02836         @param ID (int): Unique material model ID.
02837         @param section (RCRectShape): RCRectShape section object.
02838         @param calibration (str): Label of the calibration parameter set. The options are listed in
      UVCCalibrated.
02839         Defaults to 'S460NL_25mm_plate'. Change it accordingly to the steel rebars material
      properties.
02840         """
02841         self.section = deepcopy(section)
02842         super().__init__(ID, calibration, section.fy, section.Ey)
02843         self.section_name_tag = section.name_tag
02844         self.UpdateStoredData()
02845
02846
```

7.63.3 Member Data Documentation

7.63.3.1 section

```
section
```

Definition at line 2841 of file MaterialModels.py.

7.63.3.2 section_name_tag

```
section_name_tag
```

Definition at line 2843 of file MaterialModels.py.

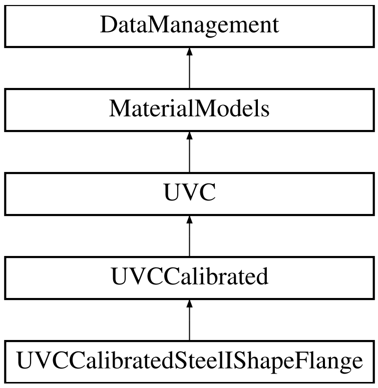
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.64 UVCCalibratedSteelIShapeFlange Class Reference

Class that is the children of UVCCalibrated and combine the class SteelIShape (section) to retrieve the information needed for the material model of the flange (often used fo the entire section).

Inheritance diagram for UVCCalibratedSteelIShapeFlange:



Public Member Functions

- def __init__ (self, int ID, SteelIShape section, calibration='S355J2_HEB500_flange')

    Constructor of the class.

Public Attributes

- section
- section_name_tag

7.64.1 Detailed Description

Class that is the children of UVCCalibrated and combine the class SteelIShape (section) to retrieve the information needed for the material model of the flange (often used fo the entire section).

Parameters

UVCCalibrated	Parent class.

Definition at line 2868 of file MaterialModels.py.

7.64.2 Constructor & Destructor Documentation

7.64.2.1 __init__()

```
def __init__ (
              self,
             int ID,
             SteelIShape section,
              calibration = 'S355J2_HEB500_flange' )
```

Constructor of the class.

Parameters

ID	(int): Unique material model ID.
section	(SteelIShape): SteelIShape section object.
calibration	(str, optional): Label of the calibration parameter set. The options are listed in UVCCalibrated. Defaults to 'S355J2_HEB500_flange'. Change it accordingly to the steel rebars material properties.

Reimplemented from UVCCalibrated.

Definition at line 2875 of file MaterialModels.py.

```
02875     def __init__(self, ID: int, section: SteelIShape, calibration = 'S355J2_HEB500_flange'):
02876         """
02877         Constructor of the class.
02878 
02879         @param ID (int): Unique material model ID.
02880         @param section (SteelIShape): SteelIShape section object.
02881         @param calibration (str, optional): Label of the calibration parameter set. The options are
       listed in UVCCalibrated.
02882             Defaults to 'S355J2_HEB500_flange'. Change it accordingly to the steel rebars material
       properties.
02883         """
02884         self.section = deepcopy(section)
02885         super().__init__(ID, calibration, section.Fy, section.E)
02886         self.section_name_tag = section.name_tag
02887         self.UpdateStoredData()
02888 
02889 
```

7.64.3 Member Data Documentation

7.64.3.1 section

```
section
```

Definition at line 2884 of file MaterialModels.py.

7.64.3.2 section_name_tag

```
section_name_tag
```

Definition at line 2886 of file MaterialModels.py.

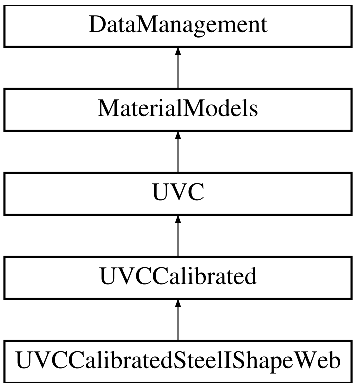
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.65 UVCCalibratedSteelIShapeWeb Class Reference

Class that is the children of UVCCalibrated and combine the class SteelIShape (section) to retrieve the information needed for the material model of the web.

Inheritance diagram for UVCCalibratedSteelIShapeWeb:



Public Member Functions

- def __init__ (self, int ID, SteelIShape section, calibration='S355J2_HEB500_web')

    Constructor of the class.

Public Attributes

- section
- section_name_tag

7.65.1 Detailed Description

Class that is the children of UVCCalibrated and combine the class SteelIShape (section) to retrieve the information needed for the material model of the web.

Parameters

UVCCalibrated	Parent class.

Definition at line 2890 of file MaterialModels.py.

7.65.2 Constructor & Destructor Documentation

7.65.2.1 __init__()

```
def __init__ (
                self,
                int ID,
                SteelIShape section,
                calibration = 'S355J2_HEB500_web' )
```

Constructor of the class.

Parameters

ID	(int): Unique material model ID.
section	(SteelIShape): SteelIShape section object.
calibration	(str, optional): Label of the calibration parameter set. The options are listed in UVCCalibrated. Defaults to 'S355J2_HEB500_web'. Change it accordingly to the steel rebars material properties.

Reimplemented from UVCCalibrated.

Definition at line 2897 of file MaterialModels.py.

```
02897     def __init__(self, ID: int, section: SteelIShape, calibration = 'S355J2_HEB500_web'):
02898         """
02899         Constructor of the class.
02900 
02901         @param ID (int): Unique material model ID.
02902         @param section (SteelIShape): SteelIShape section object.
02903         @param calibration (str, optional): Label of the calibration parameter set. The options are
       listed in UVCCalibrated.
02904             Defaults to 'S355J2_HEB500_web'. Change it accordingly to the steel rebars material
       properties.
02905         """
02906         self.section = deepcopy(section)
02907         super().__init__(ID, calibration, section.Fy_web, section.E)
02908         self.section_name_tag = section.name_tag
02909         self.UpdateStoredData()
02910 
02911 
02912 # Public functions
```

7.65.3 Member Data Documentation

7.65.3.1 section

```
section
```

Definition at line 2906 of file MaterialModels.py.

7.65.3.2 section_name_tag

```
section_name_tag
```

Definition at line 2908 of file MaterialModels.py.

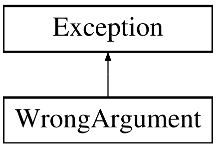
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py

7.66 WrongArgument Class Reference

Exception class for the "input of a wrong argument" error.

Inheritance diagram for WrongArgument:



7.66.1 Detailed Description

Exception class for the "input of a wrong argument" error.

Definition at line 11 of file ErrorHandling.py.

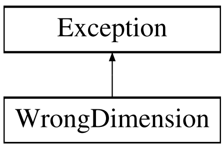
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py

7.67 WrongDimension Class Reference

Exception class for the "wrong array dimensions" error.

Inheritance diagram for WrongDimension:



7.67.1 Detailed Description

Exception class for the "wrong array dimensions" error.

Definition at line 26 of file ErrorHandling.py.

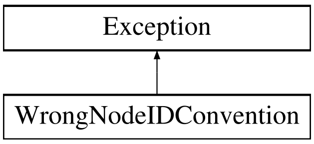
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py

7.68 WrongNodeIDConvention Class Reference

Exception class for the "wrong node ID convention definition" error.

Inheritance diagram for WrongNodeIDConvention:



Public Member Functions

- def __init__ (self, node)

Public Attributes

- node

7.68.1 Detailed Description

Exception class for the "wrong node ID convention definition" error.

Definition at line 41 of file ErrorHandling.py.

7.68.2 Constructor & Destructor Documentation

7.68.2.1 __init__()

```
def __init__ (
              self,
              node )
```

Definition at line 44 of file ErrorHandling.py.

```
00044     def __init__(self, node):
00045         self.node = node
00046
```

7.68.3 Member Data Documentation

7.68.3.1 node

`node`

Definition at line 45 of file ErrorHandling.py.

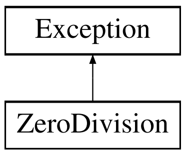
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py

7.69 ZeroDivision Class Reference

Exception class for the "zero division" error.

Inheritance diagram for ZeroDivision:



7.69.1 Detailed Description

Exception class for the "zero division" error.

Definition at line 6 of file ErrorHandling.py.

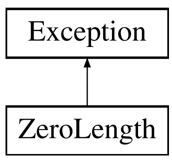
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py

7.70 ZeroLength Class Reference

Exception class for the "zero length element (non intentional)" error.

Inheritance diagram for ZeroLength:



Public Member Functions

- def __init__ (self, element)

Public Attributes

- element

7.70.1 Detailed Description

Exception class for the "zero length element (non intentional)" error.

Definition at line 52 of file ErrorHandling.py.

7.70.2 Constructor & Destructor Documentation

7.70.2.1 __init__()

```
def __init__ (
              self,
              element )
```

Definition at line 55 of file ErrorHandling.py.

```
00055     def __init__(self, element):
00056         self.element = element
```

7.70.3 Member Data Documentation

7.70.3.1 element

```
element
```

Definition at line 56 of file ErrorHandling.py.

The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py

Chapter 8

File Documentation

8.1 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Analysis↩ AndPostProcessing.py File Reference

Classes

- class Analysis

  Class dedicated to the analysis of the OpenSeesPy model.

Namespaces

- namespace AnalysisAndPostProcessing

  Module with pre-made analysis and postprocessing functions.

8.2 AnalysisAndPostProcessing.py

Go to the documentation of this file.

```
"""Module with pre-made analysis and postprocessing functions. \n
Carmine Schipani, 2021
"""

from openseespy.opensees import *
import matplotlib.pyplot as plt
import numpy as np
import os
import openseespy.postprocessing.Get_Rendering as opsplt
from OpenSeesPyAssistant.ErrorHandling import *
from OpenSeesPyAssistant.Units import *
from OpenSeesPyAssistant.Constants import *
from OpenSeesPyAssistant.FunctionalFeatures import *


class Analysis():
    """Class dedicated to the analysis of the OpenSeesPy model. The Gravity method should be run first to perform the Load-control analysis (apply the vertical load). If no vertical load, this method can be omitted. \n
    Then only one of the Displacement-control (Pushover or LoadingProtocol) or Load-control (LateralForce) analysis can ran. \n
    After the analysis reach convergence in the last step, for the postprocessing, the DeformedShape method can be used to see the final deformed shape and the animation of the entire loading protocol;
    the FiberResponse method can be used to see the animation of the same fiber section recorded during the analysis (strain and/or stress).
    """
    def __init__(self, data_dir: str,  name_ODB: str, algo = "KrylovNewton", test_type = "NormDispIncr", test_opt = 0, max_iter = MAX_ITER, tol = TOL, allow_smaller_step = False):
```

```
        """
        The constructor of the class.

        @param data_dir (str): Directory in which the results from the analysis will be stored. Use the recorders (from OpenSeesPy) or the Record method from MemberModel.
        @param name_ODB (str): Name for the folder in which the data for the animations and the fibers are stored.
        @param algo (str, optional): Type of alghoritm chosen for the analysis. It detemines how to construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the non-linear equation.
            For more information on the available types, see the OpenSeesPy documentation. Defaults to "KrylovNewton".
        @param test_type (str, optional): Type of test chosen for the analysis. It determines how to construct a ConvergenceTest object.
            Certain SolutionAlgorithm objects require a ConvergenceTest object to determine if convergence has been achieved at the end of an iteration step.
            For more information on the available types, see the OpenSeesPy documentation. Defaults to "NormDispIncr".
        @param test_opt (int, optional): Print-flag from 0 to 5 used to receive more info during the iteration
            (for example: 0 print nothing and 2 print information on norms and number of iterations at end of successful test).
            For more information, see the OpenSeesPy documentation. Defaults to 0.
        @param max_iter (float, optional): Maximal number of iterations to check. Defaults to MAX_ITER (from Constants Module).
        @param tol (float, optional): Tolerance criteria used to check for convergence. Defaults to TOL (from Constants Module).
        @param allow_smaller_step (bool, optional): Allow smaller steps in the displacement-control analysis. Defaults to False.

        @exception NegativeValue: The argument max_iter should be positive.
        @exception NegativeValue: The argument tol should be positive.
        """
        if max_iter < 0: raise NegativeValue()
        if tol < 0: raise NegativeValue()
        if not os.path.exists(data_dir):
            print("Folder {} not found in this directory; creating one".format(data_dir))
            os.makedirs(data_dir)

        self.data_dirdata_dir = data_dir
        self.name_ODBname_ODB = name_ODB
        self.algoalgo = algo
        self.test_typetest_type = test_type
        self.toltol = tol
        self.test_opttest_opt = test_opt
        self.max_itermax_iter = max_iter
        self.allow_smaller_stepallow_smaller_step = allow_smaller_step
        self.load_caseload_case = "None"


    def Gravity(self, loaded_nodes: list, Fy: list, timeSeries_ID: int, pattern_ID: int, n_step = 10, timeSeries_type = "Linear", pattern_type = "Plain",
        constraints_type = "Plain", numberer_type = "RCM", system_type = "BandGeneral", analysis_type = "Static", show_plot = False):
        """
        Method to perform the gravity analyisis with vertical loadings (load-control).
        It can be used before calling the Pushover or LoadingProtocol methods that perform the actual anlysis. If no vertical loadings present, this method can be avoided.

        @param loaded_nodes (list): List of nodes that are loaded by the the forces in Fy. The first node will be recorded (thus usually should be in the roof).
        @param Fy (list): List of vertical loadings (negative is toward the ground, thus compression; see global coordinate system).
        @param timeSeries_ID (int): ID of the timeseries.
        @param pattern_ID (int): ID of the pattern.
        @param n_step (int, optional): Number of steps used to during the analysis to reach the objective state (with 100% vertical loadings imposed). Defaults to 10.
        @param timeSeries_type (str, optional): Type of timeseries chosen.
            For more information, see the OpenSeesPy documentation. Defaults to "Linear".
        @param pattern_type (str, optional): Type of pattern chosen.
            For more information, see the OpenSeesPy documentation. Defaults to "Plain".
        @param constraints_type (str, optional): Type of contraints chosen. It detemines how the constraint equations are enforced in the analysis.
            For more information, see the OpenSeesPy documentation. Defaults to "Plain".
        @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom.
            For more information, see the OpenSeesPy documentation. Defaults to "RCM".
        @param system_type (str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis.
            For more information, see the OpenSeesPy documentation. Defaults to "BandGeneral".
        @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct the Analysis object, which defines what type of analysis is to be performed.
            For more information, see the OpenSeesPy documentation. Defaults to "Static".
        @param show_plot (bool, optional): Option to show the 'vertical displacement vs. vertical loading' curve after the analysis. Defaults to False.

```

```
        @exception WrongDimension: The dimension of the loaded_nodes and Fy arguments needs to be the same.
        @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
        @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
        """
        if len(loaded_nodes) != len(Fy): raise WrongDimension()
        if timeSeries_ID < 1: raise NegativeValue()
        if pattern_ID < 1: raise NegativeValue()

        # for mass defined: opsplt.createODB(self.name_ODB, "Gravity", Nmodes = nEigen);
        # for tracking gravity with ODB: opsplt.createODB(self.name_ODB, "Gravity");

        # Create load pattern
        timeSeries(timeSeries_type, timeSeries_ID)
        pattern(pattern_type, timeSeries_ID, pattern_ID)
        for ii, node_ID in enumerate(loaded_nodes):
            load(node_ID, 0.0, Fy[ii], 0.0)      # load(IDNode, Fx, Fy, Mz)
        DGravity = 1.0/n_step                    # load increment

        # Set up analysis options
        constraints(constraints_type)   # how it handles boundary conditions
        numberer(numberer_type)         # renumber dof's to minimize band-width (optimization)
        system(system_type)             # how to store and solve the system of equations in the analysis
                                        # For static model, BandGeneral, for transient and/or big model, UmfPack
        integrator("LoadControl", DGravity) # LoadControl and DisplacementControl only with static model, linear TimeSeries w/ factor of 1
                                            # Newmark used for transient model
        algorithm("Newton")             # placeholder
        analysis(analysis_type)         # define type of analysis: static for pushover

        # Analysis
        dataG = np.zeros((n_step+1,2))
        print("")
        print("Gravity analysis starts")
        for iteration in range(n_step):
            convergence = self.__LoadCtrlLoop__LoadCtrlLoop(DGravity, iteration,
                self.algoalgo, self.test_typetest_type, self.toltol, self.test_opttest_opt, self.max_itermax_iter)
            if convergence != 0: break
            dataG[iteration+1,0] = nodeDisp(loaded_nodes[0], 2)/mm_unit
            dataG[iteration+1,1] = getLoadFactor(pattern_ID)*Fy[0]/kN_unit

        if show_plot:
            plt.plot(dataG[:,0], dataG[:,1])
            plt.xlabel('Vertical Displacement [mm]')
            plt.ylabel('Vertical Load [kN]')
            plt.title('Gravity curve')
            plt.show()

        loadConst("-time", 0.0)

        print("")
        print("Gravity complete")


    def LateralForce(self, loaded_nodes: list, Fx: list, timeSeries_ID: int, pattern_ID: int, n_step = 1000, fiber_ID_analysed = -1, fiber_section = 1,
        timeSeries_type = "Linear", pattern_type = "Plain", constraints_type = "Plain", numberer_type = "RCM", system_type = "BandGeneral", analysis_type = "Static",
        show_plot = True, block = False):
        """
        Method to perform the lateral force analyisis with lateral loading (load-control).
        If this method is called, the LoadingProtocol and Pushover methods should be avoided.

        @param loaded_nodes (list): List of nodes that are loaded by the the forces in Fx. The first node will be recorded (thus usually should be in the roof).
        @param Fx (list): List of horizontal loadings (negative is toward left; see global coordinate system).
        @param timeSeries_ID (int): ID of the timeseries.
        @param pattern_ID (int): ID of the pattern.
        @param n_step (int, optional): Number of steps used to during the analysis to reach the objective state (with 100% horizontal loadings imposed). Defaults to 1000.
        @param fiber_ID_analysed (int, optional): The ID of the analysed fiber. If fibers are present in the model and the user wants to save ODB data
            (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the fiber chosen.
            -1 will ignore the storage of data for fibers. Defaults to -1.
        @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
            If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
        @param timeSeries_type (str, optional): Type of timeseries chosen.
            For more information, see the OpenSeesPy documentation. Defaults to "Linear".
        @param pattern_type (str, optional): Type of pattern chosen.
            For more information, see the OpenSeesPy documentation. Defaults to "Plain".
        @param constraints_type (str, optional): Type of contraints chosen. It detemines how the constraint equations are enforced in the analysis.
```

```
            For more information, see the OpenSeesPy documentation. Defaults to "Plain".
        @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom.
            For more information, see the OpenSeesPy documentation. Defaults to "RCM".
        @param system_type (str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis.
            For more information, see the OpenSeesPy documentation. Defaults to "BandGeneral".
        @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct the Analysis object, which defines what type of analysis is to be performed.
            For more information, see the OpenSeesPy documentation. Defaults to "Static".
        @param show_plot (bool, optional): Option to show the 'Horizontal displacement vs. Horizontal loading' curve after the analysis. Defaults to True.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

        @exception WrongDimension: The dimension of the loaded_nodes and Fx arguments needs to be the same.
        @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
        @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
        @exception NegativeValue: The ID of fiber_ID_analysed needs to be a positive integer.
        """
        if len(loaded_nodes) != len(Fx): raise WrongDimension()
        if timeSeries_ID < 1: raise NegativeValue()
        if pattern_ID < 1: raise NegativeValue()
        if fiber_ID_analysed != -1 and fiber_ID_analysed < 1: raise NegativeValue()

        # for mass defined: opsplt.createODB(self.name_ODB, "LateralForce", Nmodes = nEigen);
        opsplt.createODB(self.name_ODBname_ODB, "LateralForce");
        if fiber_ID_analysed != -1: opsplt.saveFiberData2D(self.name_ODBname_ODB, "LateralForce", fiber_ID_analysed, fiber_section)

        # Create load pattern
        timeSeries(timeSeries_type, timeSeries_ID)
        pattern(pattern_type, timeSeries_ID, pattern_ID)
        for ii, node_ID in enumerate(loaded_nodes):
            load(node_ID, Fx[ii], 0.0, 0.0)      # load(IDNode, Fx, Fy, Mz)
        force = 1.0/n_step                   # load increment

        # Set up analysis options
        constraints(constraints_type)   # how it handles boundary conditions
        numberer(numberer_type)         # renumber dof's to minimize band-width (optimization)
        system(system_type)             # how to store and solve the system of equations in the analysis
                                        # For static model, BandGeneral, for transient and/or big model, UmfPack
        integrator("LoadControl", force)# LoadControl and DisplacementControl only with static model, linear TimeSeries w/ factor of 1
                                        # Newmark used for transient model
        algorithm("Newton")             # placeholder
        analysis(analysis_type)         # define type of analysis: static for pushover

        # Analysis
        dataLF = np.zeros((n_step+1,2))
        print("")
        print("Lateral Force analysis starts")
        for iteration in range(n_step):
            convergence = self.__LoadCtrlLoop__LoadCtrlLoop(force, iteration,
                self.algoalgo, self.test_typetest_type, self.toltol, self.test_opttest_opt, self.max_itermax_iter)
            if convergence != 0: break
            dataLF[iteration+1,0] = nodeDisp(loaded_nodes[0], 1)/mm_unit
            dataLF[iteration+1,1] = getLoadFactor(pattern_ID)*Fx[0]/kN_unit

        if show_plot:
            plt.plot(dataLF[:,0], dataLF[:,1])
            plt.xlabel('Lateral Displacement [mm]')
            plt.ylabel('Lateral Load [kN]')
            plt.title('Lateral force curve')
            if block:
                plt.show()

        loadConst("-time", 0.0)

        print("")
        print("Lateral force complete")
        self.load_caseload_case = "LateralForce"

        wipe()


    def Pushover(self, CtrlNode: int, Dmax, Dincr, timeSeries_ID: int, pattern_ID: int, Fx = 1*kN_unit, ele_fiber_ID_analysed = -1, fiber_section = 1,
        timeSeries_type = "Linear", pattern_type = "Plain", constraints_type = "Plain", numberer_type = "RCM", system_type = "UmfPack", analysis_type = "Static",
        show_plot = True, block = False):
        """
```

```
        Method to perform a pushover analysis (displacement-control). If this method is called, the LoadingProtocol and LateralForce methods should be avoided.

        @param CtrlNode (int): The node that will be used to impose the displacement Dmax of the pushover analysis.
            If the show_plot option is True, the curve displayed follows this node.
        @param Dmax (float): The imposed displacement.
        @param Dincr (float): The incremental displacement to reach Dmax. To converge, it should be small enough (1000 times smaller of Dmax).
        @param timeSeries_ID (int): ID of the timeseries.
        @param pattern_ID (int): ID of the pattern.
        @param Fx (float, optional): The force imposed at the control node CtrlNode. It is used for convergence reasons and it can be arbitrarly small.
            Defaults to 1*kN_unit.
        @param ele_fiber_ID_analysed (int, optional): The ID of the analysed element with fibers. If fibers are present in the model and the user wants to save ODB data
            (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the element with fibers chosen.
            -1 will ignore the storage of data for fibers. Defaults to -1.
        @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
            If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
        @param timeSeries_type (str, optional): Type of timeseries chosen.
            For more information, see the OpenSeesPy documentation. Defaults to "Linear".
        @param pattern_type (str, optional): Type of pattern chosen.
            For more information, see the OpenSeesPy documentation. Defaults to "Plain".
        @param constraints_type (str, optional): Type of contraints chosen. It detemines how the constraint equations are enforced in the analysis.
            For more information, see the OpenSeesPy documentation. Defaults to "Plain".
        @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom.
            For more information, see the OpenSeesPy documentation. Defaults to "RCM".
        @param system_type (str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis.
            For more information, see the OpenSeesPy documentation. Defaults to "UmfPack".
        @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct the Analysis object, which defines what type of analysis is to be performed.
            For more information, see the OpenSeesPy documentation. Defaults to "Static".
        @param show_plot (bool, optional): Option to show the 'lateral displacement vs. lateral loading' curve after the analysis. Defaults to True.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

        @exception NegativeValue: The ID of CtrlNode needs to be a positive integer.
        @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
        @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
        @exception NegativeValue: The ID of ele_fiber_ID_analysed needs to be a positive integer if is different from -1.
        """
        if CtrlNode < 1: raise NegativeValue()
        if timeSeries_ID < 1: raise NegativeValue()
        if pattern_ID < 1: raise NegativeValue()
        if ele_fiber_ID_analysed != -1 and ele_fiber_ID_analysed < 1: raise NegativeValue()

        # for mass defined: opsplt.createODB(self.name_ODB, "Pushover", Nmodes = nEigen);
        opsplt.createODB(self.name_ODBname_ODB, "Pushover");
        if ele_fiber_ID_analysed != -1: opsplt.saveFiberData2D(self.name_ODBname_ODB, "Pushover", ele_fiber_ID_analysed, fiber_section)

        # Create load pattern
        timeSeries(timeSeries_type, timeSeries_ID)
        pattern(pattern_type, timeSeries_ID, pattern_ID)
        load(CtrlNode, Fx, 0.0, 0.0)     # load(IDNode, Fx, Fy, Mz)
        Nsteps = int(abs(Dmax/Dincr))    # number of pushover analysis steps

        # Set up analysis options
        constraints(constraints_type)    # how it handles boundary conditions
        numberer(numberer_type)          # renumber dof's to minimize band-width (optimization)
        system(system_type)              # how to store and solve the system of equations in the analysis
                                         # For static model, BandGeneral, for transient and/or big model, UmfPack
        integrator("LoadControl", 1)     # placeholder
        algorithm("Newton")              # placeholder
        analysis(analysis_type)          # define type of analysis: static for pushover

        # Analysis
        dataPO = np.zeros((Nsteps+1,2))
        next_step = 0
        print("")
        print("Pushover analysis starts")
        for iteration in range(Nsteps):
            next_step = ProgressingPercentage(Nsteps, iteration, next_step)
            convergence = self.__LatDispCtrlLoop__LatDispCtrlLoop(CtrlNode, Dincr, iteration,
                self.algoalgo, self.test_typetest_type, self.toltol, self.test_opttest_opt, self.max_itermax_iter, self.allow_smaller_stepallow_smaller_step)
            if convergence != 0: break
```

```
            dataPO[iteration+1,0] = nodeDisp(CtrlNode, 1)/mm_unit
            dataPO[iteration+1,1] = getLoadFactor(pattern_ID)*Fx/kN_unit

        if show_plot:
            plt.plot(dataPO[:,0], dataPO[:,1])
            plt.xlabel('Horizontal Displacement [mm]')
            plt.ylabel('Horizontal Load [kN]')
            plt.title('Pushover curve')
            if block:
                plt.show()

        print("")
        print("Pushover complete")
        self.load_caseload_case = "Pushover"

        wipe()


    def LoadingProtocol(self, CtrlNode: int, discr_LP: np.ndarray, timeSeries_ID: int, pattern_ID: int, Fx = 1*kN_unit, ele_fiber_ID_analysed = -1, fiber_section = 1,
        timeSeries_type = "Linear", pattern_type = "Plain", constraints_type = "Plain", numberer_type = "RCM", system_type = "UmfPack", analysis_type = "Static",
        show_plot = True, block = False):
        """
        Method to perform a loading protocol analysis (displacement-control). If this method is called, the Pushover and LateralForce methods should be avoided.

        @param CtrlNode (int): The node that will be used to impose the displacement from the discr_LP to perform the analysis.
        @param discr_LP (np.ndarray): The loading protocol array (1 dimension) discretised. It needs to be filled with imposed displacement, not SDR.
            Use the functions DiscretizeLoadProtocol and DiscretizeLinearly in FunctionalFeatures module to help create and/or discretise one.
        @param timeSeries_ID (int): ID of the timeseries.
        @param pattern_ID (int): ID of the pattern.
        @param Fx (float, optional): The force imposed at the control node CtrlNode. It is used for convergence reasons and it can be arbitrarly small.
            Defaults to 1*kN_unit.
        @param ele_fiber_ID_analysed (int, optional): The ID of the analysed element with fibers. If fibers are present in the model and the user wants to save ODB data
            (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the element with fibers chosen.
            -1 will ignore the storage of data for fibers. Defaults to -1.
        @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
            If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
        @param timeSeries_type (str, optional): Type of timeseries chosen.
            For more information, see the OpenSeesPy documentation. Defaults to "Linear".
        @param pattern_type (str, optional): Type of pattern chosen.
            For more information, see the OpenSeesPy documentation. Defaults to "Plain".
        @param constraints_type (str, optional): Type of contraints chosen. It detemines how the constraint equations are enforced in the analysis.
            For more information, see the OpenSeesPy documentation. Defaults to "Plain".
        @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom.
            For more information, see the OpenSeesPy documentation. Defaults to "RCM".
        @param system_type (str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis.
            For more information, see the OpenSeesPy documentation. Defaults to "UmfPack".
        @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct the Analysis object, which defines what type of analysis is to be performed.
            For more information, see the OpenSeesPy documentation. Defaults to "Static".
        @param show_plot (bool, optional): Option to show the 'lateral displacement vs. lateral loading' curve after the analysis. Defaults to True.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

        @exception NegativeValue: The ID of CtrlNode needs to be a positive integer.
        @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
        @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
        @exception NegativeValue: The ID of fiber_ID_analysed needs to be a positive integer if is different from -1.
        """
        if CtrlNode < 1: raise NegativeValue()
        if timeSeries_ID < 1: raise NegativeValue()
        if pattern_ID < 1: raise NegativeValue()
        if ele_fiber_ID_analysed != -1 and ele_fiber_ID_analysed < 1: raise NegativeValue()

        # for mass defined: opsplt.createODB(self.name_ODB, "LoadingProtocol", Nmodes = nEigen);
        opsplt.createODB(self.name_ODBname_ODB, "LoadingProtocol");
        if ele_fiber_ID_analysed != -1: opsplt.saveFiberData2D(self.name_ODBname_ODB, "LoadingProtocol", ele_fiber_ID_analysed, fiber_section)

        # Create load pattern
        timeSeries(timeSeries_type, timeSeries_ID)
        pattern(pattern_type, timeSeries_ID, pattern_ID)
        load(CtrlNode, Fx, 0.0, 0.0)    # load(IDNode, Fx, Fy, Mz)
```

```
        dU_prev = 0
        Nsteps = np.size(discr_LP)      # number of pushover analysis steps

        # Set up analysis options
        constraints(constraints_type)   # how it handles boundary conditions
        numberer(numberer_type)         # renumber dof's to minimize band-width (optimization)
        system(system_type)             # how to store and solve the system of equations in the analysis
                                        # For static model, BandGeneral, for transient and/or big model, UmfPack
        integrator("LoadControl", 1)    # placeholder
        algorithm("Newton")             # placeholder
        analysis(analysis_type)         # define type of analysis: static for LoadingProtocol

        # Analysis
        dataLP = np.zeros((Nsteps+1,2))
        next_step = 0
        print("")
        print("Loading Protocol analysis starts")
        for iteration in range(Nsteps):
            # Compute displacement usinf the given loading protocol (discretized)
            dU_next = discr_LP[iteration]
            dU = dU_next - dU_prev
            dU_prev = dU_next

            next_step = ProgressingPercentage(Nsteps, iteration, next_step)
            convergence = self.__LatDispCtrlLoop__LatDispCtrlLoop(CtrlNode, dU, iteration,
                self.algoalgo, self.test_typetest_type, self.toltol, self.test_opttest_opt, self.max_itermax_iter, self.allow_smaller_stepallow_smaller_step)
            if convergence != 0: break
            dataLP[iteration+1,0] = nodeDisp(CtrlNode, 1)/mm_unit
            dataLP[iteration+1,1] = getLoadFactor(pattern_ID)*Fx/kN_unit

        if show_plot:
            plt.plot(dataLP[:,0], dataLP[:,1])
            plt.xlabel('Horizontal Displacement [mm]')
            plt.ylabel('Horizontal Load [kN]')
            plt.title('Loading Protocol curve')
            if block:
                plt.show()

        print("")
        print("Loading Protocol complete")
        self.load_caseload_case = "LoadingProtocol"

        wipe()


    def __LoadCtrlLoop(self, force, iteration: int, algo = "KrylovNewton", test_type = "NormDispIncr", tol = TOL, test_opt = 0, max_iter = MAX_ITER):
        """
        PRIVATE METHOD. It is used perform one load increment 'force' load-control analysis step using 'algo' and 'test_type' as algorithm and test.
        The integrator is LoadControl. If convergence issues are encountered, the method performa a convergence analysis trying different ways to converge.

        @param force (dougle): The load increment performed.
        @param iteration (int): The current iteration.
        @param algo (str, optional): Type of alghoritm chosen for the analysis. It detemines how to construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the non-linear equation.
            For more information on the available types, see the OpenSeesPy documentation. Defaults to "KrylovNewton".
        @param test_type (str, optional): Type of test chosen for the analysis. It determines how to construct a ConvergenceTest object.
            Certain SolutionAlgorithm objects require a ConvergenceTest object to determine if convergence has been achieved at the end of an iteration step.
            For more information on the available types, see the OpenSeesPy documentation. Defaults to "NormDispIncr".
        @param tol (float, optional): Tolerance criteria used to check for convergence. Defaults to TOL (from Constants Module).
        @param test_opt (int, optional): Print-flag from 0 to 5 used to receive more info during the iteration
            (for example: 0 print nothing and 2 print information on norms and number of iterations at end of successful test).
            For more information, see the OpenSeesPy documentation. Defaults to 0.
        @param max_iter (float, optional): Maximal number of iterations to check. Defaults to MAX_ITER (from Constants Module).

        @exception NegativeValue: iteration needs to be a positive integer.
        @exception NegativeValue: tol needs to be positive.
        @exception NegativeValue: max_iter needs to be positive.

        @returns int: 0 if the interation converged.
        """
        if iteration < 0: raise NegativeValue()
        if tol < 0: raise NegativeValue()
```

```
        if max_iter < 0: raise NegativeValue()

        # Default analysis
        integrator("LoadControl", force)          # LoadControl and DisplacementControl only with static model, linear TimeSeries w/ factor of 1
                                                   # Newmark used for transient model
        test(test_type, tol, max_iter, test_opt)   # type of convergence criteria with tolerance, max iterations;
                                                   # Normally use EnergyIncr, if conv issues, try NormDispIncr; optional: test_opt = 2 for debugging
        algorithm(algo)                            # use Newton's solution algorithm: updates tangent stiffness at every iteration
        convergence = analyze(1)                   # this will return zero if no convergence problems were encountered

        # Convergence analysis
        if convergence != 0:
            print("---------------------------------------------------------------------------------------")
            print("Vertical Load-control analysis failed at iteration {}".format(iteration))
            print("Convergence analysis starts")
            print("---------------------------------------------------------------------------------------")

            if convergence != 0:
                print("Try 1")
                convergence = _ConvergenceTest("KrylovNewton", "NormDispIncr", tol, max_iter, test_opt)

            if convergence != 0:
                print("Try 2")
                convergence = _ConvergenceTest("KrylovNewton", "NormDispIncr", tol*10, max_iter*10, test_opt)

            if convergence != 0:
                print("Try 3")
                convergence = _ConvergenceTest("KrylovNewton", "EnergyIncr", tol, max_iter, test_opt)

            if convergence != 0:
                print("Try 4")
                convergence = _ConvergenceTest("KrylovNewton", "EnergyIncr", tol*10, max_iter*10, test_opt)

            if convergence != 0:
                print("Try 5")
                convergence = _ConvergenceTest("Newton", "NormDispIncr", tol, max_iter, test_opt)

            if convergence != 0:
                print("Try 6")
                convergence = _ConvergenceTest("Newton", "NormDispIncr", tol*10, max_iter*10, test_opt)

            if convergence != 0:
                print("Try 7")
                convergence = _ConvergenceTest("Newton", "EnergyIncr", tol, max_iter, test_opt)

            if convergence != 0:
                print("Try 8")
                convergence = _ConvergenceTest("Newton", "EnergyIncr", tol*10, max_iter*10, test_opt)

            if convergence != 0:
                print("Try 9")
                convergence = _ConvergenceTest("ModifiedNewton", "NormDispIncr", tol, max_iter, test_opt)

            if convergence != 0:
                print("Try 10")
                convergence = _ConvergenceTest("ModifiedNewton", "NormDispIncr", tol*10, max_iter*10, test_opt)

            if convergence != 0:
                print("Try 11")
                convergence = _ConvergenceTest("ModifiedNewton", "EnergyIncr", tol, max_iter, test_opt)

            if convergence != 0:
                print("Try 12")
                convergence = _ConvergenceTest("ModifiedNewton", "EnergyIncr", tol*10, max_iter*10, test_opt)

            if convergence != 0:
                print("")
                print("#############################################################################")
                print("NO CONVERGENCE! Load-control analysis stops at iteration {}                 ".format(iteration))
                print("#############################################################################")
                print("")
```

```
            else:
                print("Convergence reached, convergence analysis ends.")

        return convergence


    def __LatDispCtrlLoop(self, CtrlNode, dU, iteration, algo = "KrylovNewton", test_type = "NormDispIncr", tol = TOL, test_opt = 0, max_iter = MAX_ITER, allow_smaller_step = False):
        """
        PRIVATE METHOD. It is used perform one imposed displacement increment 'dU' displacement-control analysis step using 'algo' and 'test_type' as algorithm and test.
        The integrator is DisplacementControl. If convergence issues are encountered, the method performa a convergence analysis trying different ways to converge.

        @param CtrlNode ([type]): [description]
        @param dU (float): The imposed displacement increment for the current iteration
        @param iteration (int): The current iteration.
        @param algo (str, optional): Type of alghoritm chosen for the analysis. It detemines how to construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the non-linear equation.
            For more information on the available types, see the OpenSeesPy documentation. Defaults to "KrylovNewton".
        @param test_type (str, optional): Type of test chosen for the analysis. It determines how to construct a ConvergenceTest object.
            Certain SolutionAlgorithm objects require a ConvergenceTest object to determine if convergence has been achieved at the end of an iteration step.
            For more information on the available types, see the OpenSeesPy documentation. Defaults to "NormDispIncr".
        @param tol (float, optional): Tolerance criteria used to check for convergence. Defaults to TOL (from Constants Module).
        @param test_opt (int, optional): Print-flag from 0 to 5 used to receive more info during the iteration
            (for example: 0 print nothing and 2 print information on norms and number of iterations at end of successful test).
            For more information, see the OpenSeesPy documentation. Defaults to 0.
        @param max_iter (float, optional): Maximal number of iterations to check. Defaults to MAX_ITER (from Constants Module).
        @param allow_smaller_step (bool, optional): Allow smaller steps in the displacement-control analysis. Defaults to False.

        @exception NegativeValue: iteration needs to be a positive integer.
        @exception NegativeValue: tol needs to be positive.
        @exception NegativeValue: max_iter needs to be positive.

        @returns int: 0 if the interation converged.
        """
        if iteration < 0: raise NegativeValue()
        if tol < 0: raise NegativeValue()
        if max_iter < 0: raise NegativeValue()

        # Default analysis
        CtrlDOF = 1
        integrator("DisplacementControl", CtrlNode, CtrlDOF, dU, 1, dU, dU)  # use displacement-controlled analysis
        test(test_type, tol, max_iter, test_opt)     # type of convergence criteria with tolerance, max iterations;
                                                     # Normally use EnergyIncr, if conv issues, try NormDispIncr; optional: test_opt = 2 for debugging
        algorithm(algo)                              # use Newton's solution algorithm: updates tangent stiffness at every iteration
        convergence = analyze(1)                     # this will return zero if no convergence problems were encountered

        # Convergence analysis
        if convergence != 0:
            print("---------------------------------------------------------------------------------------")
            print("Lateral Displacement-control analysis failed at iteration {} and control node lateral displacement = {} mm.".format(iteration, nodeDisp(CtrlNode, CtrlDOF)/mm_unit))
            print("Convergence analysis starts")
            print("---------------------------------------------------------------------------------------")

            if convergence != 0:
                print("Try 1")
                convergence = _ConvergenceTest("KrylovNewton", "NormDispIncr", tol, max_iter, test_opt)

            if convergence != 0:
                print("Try 2")
                convergence = _ConvergenceTest("KrylovNewton", "NormDispIncr", tol*10, max_iter*10, test_opt)

            if convergence != 0:
                print("Try 3")
                convergence = _ConvergenceTest("KrylovNewton", "EnergyIncr", tol, max_iter, test_opt)

```

```
            if convergence != 0:
                print("Try 4")
                convergence = _ConvergenceTest("KrylovNewton", "EnergyIncr", tol*10, max_iter*10, test_opt)

            if convergence != 0:
                print("Try 5")
                convergence = _ConvergenceTest("Newton", "NormDispIncr", tol, max_iter, test_opt)

            if convergence != 0:
                print("Try 6")
                convergence = _ConvergenceTest("Newton", "NormDispIncr", tol*10, max_iter*10, test_opt)

            if convergence != 0:
                print("Try 7")
                convergence = _ConvergenceTest("Newton", "EnergyIncr", tol, max_iter, test_opt)

            if convergence != 0:
                print("Try 8")
                convergence = _ConvergenceTest("Newton", "EnergyIncr", tol*10, max_iter*10, test_opt)

            if convergence != 0:
                print("Try 9")
                convergence = _ConvergenceTest("ModifiedNewton", "NormDispIncr", tol, max_iter, test_opt)

            if convergence != 0:
                print("Try 10")
                convergence = _ConvergenceTest("ModifiedNewton", "NormDispIncr", tol*10, max_iter*10, test_opt)

            if convergence != 0:
                print("Try 11")
                convergence = _ConvergenceTest("ModifiedNewton", "EnergyIncr", tol, max_iter, test_opt)

            if convergence != 0:
                print("Try 12")
                convergence = _ConvergenceTest("ModifiedNewton", "EnergyIncr", tol*10, max_iter*10, test_opt)

            if convergence != 0 and allow_smaller_step:
                print("Use smaller steps")
                print("10 times more intergator iteration, min_dU = dU/10")
                integrator("DisplacementControl", CtrlNode, CtrlDOF, dU, 10, dU/10, dU)

                if convergence != 0:
                    print("Try 13")
                    convergence = _ConvergenceTest("KrylovNewton", "NormDispIncr", tol, max_iter, test_opt)

                if convergence != 0:
                    print("Try 14")
                    convergence = _ConvergenceTest("KrylovNewton", "EnergyIncr", tol, max_iter, test_opt)

                if convergence != 0:
                    print("Try 15")
                    convergence = _ConvergenceTest("Newton", "NormDispIncr", tol, max_iter, test_opt)

                if convergence != 0:
                    print("Try 16")
                    convergence = _ConvergenceTest("Newton", "EnergyIncr", tol, max_iter, test_opt)

                if convergence != 0:
                    print("Try 17")
                    convergence = _ConvergenceTest("ModifiedNewton", "NormDispIncr", tol, max_iter, test_opt)

                if convergence != 0:
                    print("Try 19")
                    convergence = _ConvergenceTest("ModifiedNewton", "EnergyIncr", tol, max_iter, test_opt)

            if convergence != 0:
                print("")
                print("############################################################################")
                print("NO CONVERGENCE! Lateral Displacement-control analysis stops at iteration {} and control node lateral displacement = {} mm.".format(iteration, nodeDisp(CtrlNode, CtrlDOF)/mm_unit))
                print("############################################################################")
                print("")
            else:
                print("Convergence reached, convergence analysis ends.")

        return convergence

```

```

    def DeformedShape(self, scale = 1, animate = False, dt = 0.01):
        """
        Method that shows the final deformed shape of the model. It can also show the animation that shows how the model behaved during the analysis.

        @param scale (int, optional): The scaling factor to magnify the deformation. The value should be adjusted for each model. Defaults to 1.
        @param animate (bool, optional): Option to show the animation of the model during the analysis. Defaults to False.
        @param dt (float, optional): The time step between every iteration. Defaults to 0.01.

        @exception NameError: The methods for the analysis were not called.
        """
        if self.load_caseload_case == "None": raise NameError("The analysis is not complete.")

        # Display deformed shape, the scaling factor needs to be adjusted for each model
        opsplt.plot_deformedshape(Model = self.name_ODBname_ODB, LoadCase=self.load_caseload_case, scale = scale)
        if animate:
            opsplt.animate_deformedshape(Model = self.name_ODBname_ODB, LoadCase=self.load_caseload_case, dt = dt, scale = scale)


    def FiberResponse(self, ele_fiber_ID_analysed: int, fiber_section = 1, animate_stress = False, animate_strain = False, fps = 25):
        """
        Method that shows the final stress response of the fiber section chosen.
        It can also show the animation that shows how the fiber section behaved during the analysis. The fiber ID and section needs to be recorded during the analysis,
        thus if the method LateralForce, Pushover or LoadingProtocol was used, the same fiber ID and section need to be used.

        @param ele_fiber_ID_analysed (int): The ID of the analysed fiber. If fibers are present in the model and the user wants to save ODB data
            (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the fiber chosen.
            -1 will ignore the storage of data for fibers.
        @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
            If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
        @param animate_stress (bool, optional): Option to show the animation of the fiber stress during the analysis. Defaults to False.
        @param animate_strain (bool, optional): Option to show the animation of the fiber strain during the analysis. Defaults to False.
        @param fps (int, optional): Number of frame per seconds for the animations. Defaults to 25.
        @exception NameError: The methods for the analysis were not called.
        """
        if self.load_caseload_case == "None": raise NameError("The analysis is not complete.")

        opsplt.plot_fiberResponse2D(self.name_ODBname_ODB, self.load_caseload_case, ele_fiber_ID_analysed, fiber_section, InputType = 'stress')
        if animate_stress:
            ani1 = opsplt.animate_fiberResponse2D(self.name_ODBname_ODB, self.load_caseload_case, ele_fiber_ID_analysed, fiber_section, InputType = 'stress', fps = fps)
        if animate_strain:
            ani1 = opsplt.animate_fiberResponse2D(self.name_ODBname_ODB, self.load_caseload_case, ele_fiber_ID_analysed, fiber_section, InputType = 'strain', fps = fps)


def _ConvergenceTest(algo_type: str, test_type: str, tol, max_iter, test_opt = 0):
    """
    PRIVATE FUNCTION. It is used during the convergence analysis to test different ways to reach convergence.

    @param algo_type (str): Type of alghoritm chosen for the analysis. It detemines how to construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the non-linear equation.
        For more information on the available types, see the OpenSeesPy documentation.
    @param test_type (str): Type of test chosen for the analysis. It determines how to construct a ConvergenceTest object.
        Certain SolutionAlgorithm objects require a ConvergenceTest object to determine if convergence has been achieved at the end of an iteration step.
        For more information on the available types, see the OpenSeesPy documentation.
    @param tol (float): Tolerance criteria used to check for convergence.
    @param max_iter (float): Maximal number of iterations to check.
    @param test_opt (int, optional): Print-flag from 0 to 5 used to receive more info during the iteration
        (for example: 0 print nothing and 2 print information on norms and number of iterations at end of successful test).
        For more information, see the OpenSeesPy documentation. Defaults to 0.

    @returns int: 0 if the interation converged.
    """
    print("algorithm: {}".format(algo_type))
    print("test: {}, tol = {}, max iter = {}".format(test_type, tol, max_iter))
    test(test_type, tol, max_iter, test_opt)
    algorithm(algo_type)
```

```
    return analyze(1)
```

8.3 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Connections.py File Reference

Namespaces

- namespace Connections

  Module with different functions useful when defining boundary conditions (fix) or connections (pin, rigid or springs).

Functions

- def Pin (int NodeRID, int NodeCID)

  Function that constrains the translational DOF with a multi-point constraint.
- def RigidSupport (int NodeID)

  Function that fixes the x, y movements and the rotation of one node.
- def RotationalSpring (int ElementID, int NodeRID, int NodeCID, int MatID, Rigid=False)

  Function that defines a zero-length spring and constrains the translations DOFs of the spring.

8.4 Connections.py

Go to the documentation of this file.

```
"""Module with different functions useful when defining boundary conditions (fix) or connections (pin, rigid or springs). \n
Carmine Schipani, 2021
"""

from openseespy.opensees import *
from OpenSeesPyAssistant.ErrorHandling import *


def RigidSupport(NodeID: int):
    """
    Function that fixes the x, y movements and the rotation of one node.

    @param NodeID (int): ID of the node to be fixed

    @exception NegativeValue: The ID of NodeID needs to be a positive integer.
    """
    if NodeID < 1: raise NegativeValue()

    fix(NodeID, 1, 1, 1)


def Pin(NodeRID: int, NodeCID: int):
    """
    Function that constrains the translational DOF with a multi-point constraint.

    @param NodeRID (int): Node ID which will be retained by the multi-point constraint
    @param NodeCID (int): Node ID which will be constrained by the multi-point constraint

    @exception WrongArgument: The IDs passed needs to be different.
    @exception NegativeValue: The ID of NodeRID needs to be a positive integer.
    @exception NegativeValue: The ID of NodeCID needs to be a positive integer.
    """
    if NodeCID == NodeRID: raise WrongArgument()
    if NodeRID < 1: raise NegativeValue()
    if NodeCID < 1: raise NegativeValue()

    # Constrain the translational DOF with a multi-point constraint
```

```
    #           retained constrained DOF_1 DOF_2
    equalDOF(NodeRID, NodeCID, 1, 2)


def RotationalSpring(ElementID: int, NodeRID: int, NodeCID: int, MatID: int, Rigid = False):
    """
    Function that defines a zero-length spring and constrains the translations DOFs of the spring. Can be used also to create rigid connections.

    @param ElementID (int): ID of the zerolength element that models the spring
    @param NodeRID (int): Node ID which will be retained by the multi-point constraint
    @param NodeCID (int): Node ID which will be constrained by the multi-point constraint
    @param MatID (int): ID of the material model chosen
    @param Rigid (bool, optional): Optional argument that transforms the joint in a completely rigid connection. Defaults to False.

    @exception NegativeValue: The ID of ElementID needs to be a positive integer.
    @exception NegativeValue: The ID of NodeCID needs to be a positive integer.
    @exception NegativeValue: The ID of NodeRID needs to be a positive integer.
    @exception WrongArgument: The IDs of the nodes passed needs to be different.
    @exception NegativeValue: The ID of MatID needs to be a positive integer.
    """
    if ElementID < 1: raise NegativeValue()
    if NodeCID < 1: raise NegativeValue()
    if NodeRID < 1: raise NegativeValue()
    if NodeCID == NodeRID: raise WrongArgument()
    if MatID < 1: raise NegativeValue()

    if not Rigid:
        # Zero length element (spring)
        element("zeroLength", ElementID, NodeRID, NodeCID, "-mat", MatID, "-dir", 6)

        # Constrain the translational DOF with a multi-point constraint
        Pin(NodeRID, NodeCID)
    else:
        equalDOF(NodeRID, NodeCID, 1, 2, 3)
```

8.5 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/↩ Constants.py File Reference

Namespaces

- namespace Constants

  Module with the values of a set of essential constants.

Variables

- float G_CONST = 9.810∗m_unit/s_unit∗∗2
- int MAX_ITER = 100
- int MAX_ITER_INTEGRATION = 50
- float RIGID = 100.0
- float TOL = 1.0e-6
- float TOL_INTEGRATION = 1.0e-12
- float ZERO = 1.0e-9

8.6 Constants.py

Go to the documentation of this file.

```
"""Module with the values of a set of essential constants. They are consistent with the units defined in Units. \n
Carmine Schipani, 2021
"""

from OpenSeesPyAssistant.Units import *


TOL = 1.0e-6
TOL_INTEGRATION = 1.0e-12
ZERO = 1.0e-9                   # used when defining mass that is equal to 0 (avoid convergence problem)
G_CONST = 9.810*m_unit/s_unit**2
RIGID = 100.0                   # multiply with a mechanical or geometrical value to have the corresponding infinitely rigid value
MAX_ITER = 100
MAX_ITER_INTEGRATION = 50
```

8.7 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/DataManagement.py File Reference

Classes

- class DataManagement

  Abstract parent class for data management.

Namespaces

- namespace DataManagement

  Module with the parent abstract class DataManagement.

8.8 DataManagement.py

Go to the documentation of this file.

```
"""
Module with the parent abstract class DataManagement. \n
Carmine Schipani, 2021
"""

from abc import ABC, abstractmethod
from OpenSeesPyAssistant.ErrorHandling import *
import numpy as np


class DataManagement(ABC):
    """
    Abstract parent class for data management.
    Using the associated MATLAB class \n
    LOAD_CLASS.m \n
    for the postprocessing in MATLAB, allowing for simpler and more reliable data management because the parameters
    from the OpenSeesPy analysis are imported automatically.
    """

    def SaveData(self, f):
        """
        Function that lists in the command window and saves in a opened file text "f" the data from the "self" class that calls it.
        Example: call this function after this line: \n
        with open(FileName, 'w') as f:

        @param f (io.TextIOWrapper): Opened file to write into
```

```

        @exception WrongDimension: The number of lists in the list self.data needs to be 2
        """
        if len(self.data[0]) != 2: raise WrongDimension()

        delimiter = "##############################" # 30 times #
        col_delimiter = "\t" # tab
        for data_line in self.data:
            f.write('\n')
            for col in data_line:
                if type(col) == np.ndarray:
                    tmp_str = np.array_str(col, max_line_width = np.inf)
                else:
                    tmp_str = str(col)
                f.write(tmp_str)
                f.write(col_delimiter)
        f.write('\n')
        f.write('NEW INFO SECTION DELIMITER \t')
        f.write(delimiter)

    @abstractmethod
    def ShowInfo(self):
        """
        Abstract method that shows the data stored in the class in the command window.
        In some cases, it's possible to plot some information (for example the curve of the material model).
        """
        pass

    @abstractmethod
    def ReInit(self):
        """
        Abstract method that computes the value of the parameters with respect of the arguments. \n
        Use after changing the value of argument inside the class (to update the values accordingly). \n
        This function can be very useful in combination with the function "deepcopy()" from the module "copy". \n
        Be careful that the parameter self.Initialized is also copied, thus it is safer to copy the class before the method that calls the actual OpenSees commands (and initialise the object).
        """
        pass

    @abstractmethod
    def UpdateStoredData(self):
        """
        Abstract method used to define and update the self.data member variable. \n
        This member variable (self.data) is a list of lists with 2 entries (info_name and info_value)
            and for each list is stored a different member variable of the class. \n
        Useful to debug the model, export data, copy object.
        """
        pass
```

8.9 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py File Reference

Classes

- class InconsistentGeometry

  Exception class for the "inconsistent geometry" error.
- class MemberFailure

  Exception class for the "member failure" error.
- class NegativeValue

  Exception class for the "negative value (argument or result)" error.
- class NoApplicability

  Exception class for the "no applicability of formula of theory" error.
- class PositiveValue

  Exception class for the "positive value (argument or result)" error.
- class WrongArgument

  Exception class for the "input of a wrong argument" error.
- class WrongDimension

Exception class for the "wrong array dimensions" error.

- class WrongNodeIDConvention

  Exception class for the "wrong node ID convention definition" error.
- class ZeroDivision

  Exception class for the "zero division" error.
- class ZeroLength

  Exception class for the "zero length element (non intentional)" error.

Namespaces

- namespace ErrorHandling

  Module dedicated to the error handling.

8.10 ErrorHandling.py

Go to the documentation of this file.

```
"""Module dedicated to the error handling. \n
Carmine Schipani, 2021
"""


class ZeroDivision(Exception):
    """Exception class for the "zero division" error.
    """
    pass

class WrongArgument(Exception):
    """Exception class for the "input of a wrong argument" error.
    """
    pass

class NegativeValue(Exception):
    """Exception class for the "negative value (argument or result)" error.
    """
    pass

class PositiveValue(Exception):
    """Exception class for the "positive value (argument or result)" error.
    """
    pass

class WrongDimension(Exception):
    """Exception class for the "wrong array dimensions" error.
    """
    pass

class InconsistentGeometry(Exception):
    """Exception class for the "inconsistent geometry" error.
    """
    pass

class MemberFailure(Exception):
    """Exception class for the "member failure" error.
    """
    pass

class WrongNodeIDConvention(Exception):
    """Exception class for the "wrong node ID convention definition" error.
    """
    def __init__(self, node):
        self.nodenode = node

class NoApplicability(Exception):
    """Exception class for the "no applicability of formula of theory" error.
    """
    pass

class ZeroLength(Exception):
    """Exception class for the "zero length element (non intentional)" error.
    """
    def __init__(self, element):
        self.elementelement = element
```

8.11 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers.py File Reference

Classes

- class Fibers

  Parent abstract class for the storage and manipulation of a fiber's information (mechanical and geometrical parameters, etc) and initialisation in the model.
- class FibersCirc

  Class that stores funcions, material properties, geometric and mechanical parameters for a circular RC fiber section.
- class FibersCircRCCircShape

  Class that is the children of FibersCirc and combine the class RCCircShape (section) to retrieve the information needed.
- class FibersIShape

  Class that stores funcions, material properties, geometric and mechanical parameters for a steel I shape (non double symmetric) fiber section.
- class FibersIShapeSteelIShape

  Class that is the children of FibersIShape and combine the class SteelIShape (section) to retrieve the information needed.
- class FibersRect

  Class that stores funcions, material properties, geometric and mechanical parameters for a rectangular RC fiber section.
- class FibersRectRCRectShape

  Class that is the children of FibersRect and combine the class RCRectShape (section) to retrieve the information needed.

Namespaces

- namespace Fibers

  Module for the fibers (rectangular, circular and I shape).

Functions

- def create_fiber_section (fiber_info)

  Initialise fiber cross-section with OpenSeesPy commands.
- def plot_fiber_section (fiber_info, fill_shapes=True, matcolor=['#808080', '#D3D3D3', 'r', 'b', 'g', 'y'])

  Plot fiber cross-section.

8.12 Fibers.py

Go to the documentation of this file.

```
"""
Module for the fibers (rectangular, circular and I shape).
Carmine Schipani, 2021
"""

from openseespy.opensees import *
import matplotlib.pyplot as plt
from matplotlib.patches import Circle, Polygon, Wedge
import numpy as np
from copy import copy, deepcopy
from OpenSeesPyAssistant.Section import *
from OpenSeesPyAssistant.DataManagement import *
```

```
from OpenSeesPyAssistant.ErrorHandling import *
from OpenSeesPyAssistant.Units import *
from OpenSeesPyAssistant.MaterialModels import *


class Fibers(DataManagement):
    """
    Parent abstract class for the storage and manipulation of a fiber's information (mechanical
        and geometrical parameters, etc) and initialisation in the model.

    @param DataManagement: Parent abstract class.
    """
    pass


class FibersRect(Fibers):
    """
    Class that stores funcions, material properties, geometric and mechanical parameters for a rectangular RC fiber section.
    Coordinates: plotting coordinte (x, y) = fiber section coordinate (z, y) = (-x, y). For more information, see the OpenSeesPy documentation.

    @param Fibers: Parent abstract class.
    """
    def __init__(self, ID: int, b, d, Ay, D_hoops, e, unconf_mat_ID: int, conf_mat_ID: int, bars_mat_ID: int,
        bars_x: np.ndarray, ranges_y: np.ndarray, discr_core: list, discr_cover_lateral: list, discr_cover_topbottom: list, GJ = 0.0):
        """
        Constructor of the class.

        @param ID (int): Unique fiber section ID.
        @param b (float): Width of the section.
        @param d (float): Depth of the section.
        @param Ay (float): Area of one vertical reinforcing bar.
        @param D_hoops (float): Diameter of the hoops.
        @param e (float): Concrete cover.
        @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined fibers.
        @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
        @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars fibers.
        @param bars_x (np.ndarray): Array with a range of aligned vertical reinforcing bars for each row in x direction.
            Distances from border to bar centerline, bar to bar centerlines and finally bar centerline to border in the x direction (aligned).
            Starting from the left to right, from the top range to the bottom one.
            The number of bars for each range can vary; in this case, add this argument when defining the array " dtype = object"
        @param ranges_y (np.ndarray): Array of dimension 1 with the position or spacing in y of the ranges in bars_x.
            Distances from border to range centerlines, range to range centerlines and finally range centerline to border in the y direction.
            Starting from the top range to the bottom one.
        @param discr_core (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the confined core.
        @param discr_cover_lateral (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the lateral unconfined cover.
        @param discr_cover_topbottom (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top and bottom unconfined cover.
        @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: b needs to be positive.
        @exception NegativeValue: d needs to be positive.
        @exception NegativeValue: Ay needs to be positive.
        @exception NegativeValue: D_hoops needs to be positive.
        @exception NegativeValue: e needs to be positive.
        @exception NegativeValue: unconf_mat_ID needs to be a positive integer.
        @exception NegativeValue: conf_mat_ID needs to be a positive integer.
        @exception NegativeValue: bars_mat_ID needs to be a positive integer.
        @exception WrongDimension: Number of rows in the list bars_x needs to be the same of the length of ranges_y - 1.
        @exception InconsistentGeometry: The sum of the distances for each row in bars_x should be equal to the section's width (tol = 5 mm).
        @exception InconsistentGeometry: The sum of the distances in ranges_y should be equal to the section's depth (tol = 5 mm).
        @exception InconsistentGeometry: e should be smaller than half the depth and the width of the section.
        @exception WrongDimension: discr_core has a length of 2.
        @exception WrongDimension: discr_cover_lateral has a length of 2.
        @exception WrongDimension: discr_cover_topbottom has a length of 2.
        @exception NegativeValue: GJ needs to be positive.
        """
        # Check
        if ID < 1: raise NegativeValue()
```

```
        if b < 0: raise NegativeValue()
        if d < 0: raise NegativeValue()
        if Ay < 0: raise NegativeValue()
        if D_hoops < 0: raise NegativeValue()
        if e < 0: raise NegativeValue()
        if unconf_mat_ID < 1: raise NegativeValue()
        if conf_mat_ID < 1: raise NegativeValue()
        if bars_mat_ID < 1: raise NegativeValue()
        if np.size(bars_x) != np.size(ranges_y)-1: raise WrongDimension()
        geometry_tol = 5*mm_unit
        for bars in bars_x:
            if abs(np.sum(bars) - b) > geometry_tol: raise InconsistentGeometry()
        if abs(np.sum(ranges_y) - d) > geometry_tol: raise InconsistentGeometry()
        if e > b/2 or e > d/2: raise InconsistentGeometry()
        if len(discr_core) != 2: raise WrongDimension()
        if len(discr_cover_lateral) != 2: raise WrongDimension()
        if len(discr_cover_topbottom) != 2: raise WrongDimension()
        if GJ < 0: raise NegativeValue()

        # Arguments
        self.IDID = ID
        self.bb = b
        self.dd = d
        self.AyAy = Ay
        self.D_hoopsD_hoops = D_hoops
        self.ee = e
        self.unconf_mat_IDunconf_mat_ID = unconf_mat_ID
        self.conf_mat_IDconf_mat_ID = conf_mat_ID
        self.bars_mat_IDbars_mat_ID = bars_mat_ID
        self.bars_xbars_x = deepcopy(bars_x)
        self.ranges_yranges_y = copy(ranges_y)
        self.discr_corediscr_core = copy(discr_core)
        self.discr_cover_lateraldiscr_cover_lateral = copy(discr_cover_lateral)
        self.discr_cover_topbottomdiscr_cover_topbottom = copy(discr_cover_topbottom)
        self.GJGJ = GJ

        # Initialized the parameters that are dependent from others
        self.section_name_tagsection_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit()

    def ReInit(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        # Memebers
        if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
    self.section_name_tagsection_name_tag + " (modified)"

        # Parameters
        z1 = self.bb/2
        y1 = self.dd/2
        zc = z1-self.ee-self.D_hoopsD_hoops/2
        yc = y1-self.ee-self.D_hoopsD_hoops/2

        # Create the concrete core fibers
        core = [-yc, -zc, yc, zc]
        core_cmd = ['patch', 'rect', self.conf_mat_IDconf_mat_ID, *self.discr_corediscr_core, *core]

        # Create the concrete cover fibers (bottom left, top right)
        cover_up = [yc, -z1, y1, z1]
        cover_down = [-y1, -z1, -yc, z1]
        cover_left = [-yc, zc, yc, z1]
        cover_right = [-yc, -z1, yc, -zc]
        cover_up_cmd = ['patch', 'rect', self.unconf_mat_IDunconf_mat_ID,
    *self.discr_cover_topbottomdiscr_cover_topbottom, *cover_up]
        cover_down_cmd = ['patch', 'rect', self.unconf_mat_IDunconf_mat_ID,
    *self.discr_cover_topbottomdiscr_cover_topbottom, *cover_down]
        cover_left_cmd = ['patch', 'rect', self.unconf_mat_IDunconf_mat_ID,
    *self.discr_cover_lateraldiscr_cover_lateral, *cover_left]
        cover_right_cmd = ['patch', 'rect', self.unconf_mat_IDunconf_mat_ID,
    *self.discr_cover_lateraldiscr_cover_lateral, *cover_right]
        self.fib_secfib_sec = [['section', 'Fiber', self.IDID, '-GJ', self.GJGJ],
            core_cmd, cover_up_cmd, cover_down_cmd, cover_left_cmd, cover_right_cmd]

        # Create the reinforcing fibers (top, middle, bottom)
        # NB: note the order of definition of bars_x and ranges_y
        nr_bars = 0
        for range in self.bars_xbars_x:
            nr_bars += np.size(range)-1
        rebarY = -np.cumsum(self.ranges_yranges_y[0:-1]) + y1
        self.rebarYZrebarYZ = np.zeros((nr_bars, 2))

        iter = 0
        for ii, Y in enumerate(rebarY):
            rebarZ = -np.cumsum(self.bars_xbars_x[ii][0:-1]) + z1
```

```
            for Z in rebarZ:
                self.rebarYZrebarYZ[iter, :] = [Y, Z]
                iter = iter + 1

        for YZ in self.rebarYZrebarYZ:
            self.fib_secfib_sec.append(['layer', 'bar', self.bars_mat_IDbars_mat_ID, self.AyAy, *YZ])

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    # Methods
    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "FibersRect"], # Tag for differentiating different data
            ["ID", self.IDID],
            ["section_name_tag", self.section_name_tagsection_name_tag],
            ["b", self.bb],
            ["d", self.dd],
            ["Ay", self.AyAy],
            ["D_hoops", self.D_hoopsD_hoops],
            ["e", self.ee],
            ["GJ", self.GJGJ],
            ["conf_mat_ID", self.conf_mat_IDconf_mat_ID],
            ["discr_core", self.discr_corediscr_core],
            ["unconf_mat_ID", self.unconf_mat_IDunconf_mat_ID],
            ["discr_cover_topbottom", self.discr_cover_topbottomdiscr_cover_topbottom],
            ["discr_cover_lateral", self.discr_cover_lateraldiscr_cover_lateral],
            ["bars_mat_ID", self.bars_mat_IDbars_mat_ID],
            ["bars_x", self.bars_xbars_x],
            ["ranges_y", self.ranges_yranges_y],
            ["Initialized", self.InitializedInitialized]]

    def ShowInfo(self, plot = False, block = False):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the fiber. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        """
        print("")
        print("Requested info for FibersRect, ID = {}".format(self.IDID))
        print("Section associated: {} ".format(self.section_name_tagsection_name_tag))
        print("Base b = {} mm and depth d = {} mm".format(self.bb/mm_unit, self.dd/mm_unit))
        print("Confined material model ID = {}".format(self.conf_mat_IDconf_mat_ID))
        print("Unconfined material model ID = {}".format(self.unconf_mat_IDunconf_mat_ID))
        print("Bars material model ID = {}".format(self.bars_mat_IDbars_mat_ID))
        print("Discretisation in the core [IJ or x/z dir, JK or y dir] = {}".format(self.discr_corediscr_core))
        print("Discretisation in the lateral covers [IJ or x/z dir, JK or y dir] = {}".format(self.discr_cover_lateraldiscr_cover_lateral))
        print("Discretisation in the top and bottom covers [IJ or x/z dir, JK or y dir] = {}".format(self.discr_cover_topbottomdiscr_cover_topbottom))
        print("")

        if plot:
            plot_fiber_section(self.fib_secfib_sec, matcolor=['#808080', '#D3D3D3', 'k'])

            if block:
                plt.show()


    def CreateFibers(self):
        """
        Method that initialises the fiber by calling the OpenSeesPy commands.
        """
        create_fiber_section(self.fib_secfib_sec)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


class FibersRectRCRectShape(FibersRect):
    """
    Class that is the children of FibersRect and combine the class RCRectShape (section) to retrieve the information needed.

    @param FibersRect: Parent class.
    """
    def __init__(self, ID: int, section: RCRectShape, unconf_mat_ID: int, conf_mat_ID: int, bars_mat_ID: int,
        discr_core: list, discr_cover_lateral: list, discr_cover_topbottom: list, GJ=0):
        """
```

```
        Constructor of the class.

        @param ID (int): Unique fiber section ID.
        @param section (RCRectShape): RCRectShape section object.
        @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined fibers.
        @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
        @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars fibers.
        @param discr_core (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the confined core.
        @param discr_cover_lateral (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the lateral unconfined core.
        @param discr_cover_topbottom (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top and bottom unconfined core.
        @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.
        """
        self.sectionsection = deepcopy(section)
        super().__init__(ID, section.b, section.d, section.Ay, section.D_hoops, section.e, unconf_mat_ID, conf_mat_ID, bars_mat_ID,
            section.bars_position_x, section.bars_ranges_position_y, discr_core, discr_cover_lateral, discr_cover_topbottom, GJ=GJ)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()


class FibersCirc(Fibers):
    """
    Class that stores funcions, material properties, geometric and mechanical parameters for a circular RC fiber section.
    Coordinates: plotting coordinte (x, y) = fiber section coordinate (z, y) = (-x, y). For more information, see the OpenSeesPy documentation.

    @param Fibers: Parent abstract class.
    """
    def __init__(self, ID: int, b, e, D_bars, Ay, n_bars, D_hoops, unconf_mat_ID: int, conf_mat_ID: int, bars_mat_ID: int,
        discr_core: list, discr_cover: list, alpha_i = 0.0, GJ = 0.0):
        """
        Constructor of the class.

        @param ID (int): Unique fiber section ID.
        @param b (float): Width of the section.
        @param e (float): Concrete cover.
        @param D_bars (float): Diameter of vertical reinforcing bars.
        @param Ay (float): Area of one vertical reinforcing bar.
        @param n_bars (float): Number of reinforcement (allow float for computing the equivalent n_bars with different reinforcement areas).
        @param D_hoops (float): Diameter of the hoops.
        @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined fibers.
        @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
        @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars fibers.
        @param discr_core (list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges),
            number of subdivisions (fibers) in the radial direction (number of rings) for the confined core.
        @param discr_cover (list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges),
            number of subdivisions (fibers) in the radial direction (number of rings) for the unconfined cover.
        @param alpha_i (float, optional): Angle in deg of the first vertical rebars with respect to the y axis, counterclockwise. Defaults to 0.0.
        @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: b needs to be positive.
        @exception NegativeValue: e needs to be positive.
        @exception InconsistentGeometry: e can't be bigger than half of the width b.
        @exception NegativeValue: D_bars needs to be positive.
        @exception NegativeValue: Ay needs to be positive.
        @exception NegativeValue: n_bars needs to be positive.
        @exception NegativeValue: D_hoops needs to be positive.
        @exception NegativeValue: unconf_mat_ID needs to be a positive integer.
        @exception NegativeValue: conf_mat_ID needs to be a positive integer.
        @exception NegativeValue: bars_mat_ID needs to be a positive integer.
        @exception WrongDimension: discr_core has a length of 2.
        @exception WrongDimension: discr_cover has a length of 2.
        @exception NegativeValue: GJ needs to be positive.
        """
        # Check
        if ID < 1: raise NegativeValue()
        if b < 0: raise NegativeValue()
        if e < 0: raise NegativeValue()
```

```
        if e > b/2: raise InconsistentGeometry()
        if D_bars < 0: raise NegativeValue()
        if Ay < 0: raise NegativeValue()
        if n_bars < 0: raise NegativeValue()
        if D_hoops < 0: raise NegativeValue()
        if unconf_mat_ID < 1: raise NegativeValue()
        if conf_mat_ID < 1: raise NegativeValue()
        if bars_mat_ID < 1: raise NegativeValue()
        if len(discr_core) != 2: raise WrongDimension()
        if len(discr_cover) != 2: raise WrongDimension()
        if GJ < 0: raise NegativeValue()

        # Arguments
        self.IDID = ID
        self.bb = b
        self.ee = e
        self.D_barsD_bars = D_bars
        self.AyAy = Ay
        self.n_barsn_bars = n_bars
        self.D_hoopsD_hoops = D_hoops
        self.unconf_mat_IDunconf_mat_ID = unconf_mat_ID
        self.conf_mat_IDconf_mat_ID = conf_mat_ID
        self.bars_mat_IDbars_mat_ID = bars_mat_ID
        self.discr_corediscr_core = copy(discr_core)
        self.discr_coverdiscr_cover = copy(discr_cover)
        self.alpha_ialpha_i = alpha_i
        self.GJGJ = GJ

        # Initialized the parameters that are dependent from others
        self.section_name_tagsection_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit()

    def ReInit(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        # Memebers
        if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag = self.section_name_tagsection_name_tag + " (modified)"

        # Parameters
        self.r_barsr_bars = self.bb/2 - self.ee - self.D_hoopsD_hoops - self.D_barsD_bars/2
        self.r_corer_core = self.bb/2 - self.ee - self.D_hoopsD_hoops/2

        # Create the concrete core fibers
        core_cmd = ['patch', 'circ', self.conf_mat_IDconf_mat_ID, *self.discr_corediscr_core, 0, 0, 0, self.r_corer_core]

        # Create the concrete cover fibers
        cover_cmd = ['patch', 'circ', self.unconf_mat_IDunconf_mat_ID, *self.discr_coverdiscr_cover, 0, 0, self.r_corer_core, self.bb/2]
        self.fib_secfib_sec = [['section', 'Fiber', self.IDID, '-GJ', self.GJGJ],
            core_cmd, cover_cmd]

        # Create the reinforcing fibers
        bars_cmd = ['layer', 'circ', self.bars_mat_IDbars_mat_ID, self.n_barsn_bars, self.AyAy, 0, 0, self.r_barsr_bars, self.alpha_ialpha_i]
        self.fib_secfib_sec.append(bars_cmd)

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    # Methods
    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "FibersCirc"], # Tag for differentiating different data
            ["ID", self.IDID],
            ["section_name_tag", self.section_name_tagsection_name_tag],
            ["b", self.bb],
            ["e", self.ee],
            ["r_core", self.r_corer_core],
            ["D_bars", self.D_barsD_bars],
            ["Ay", self.AyAy],
            ["n_bars", self.n_barsn_bars],
            ["r_bars", self.r_barsr_bars],
            ["D_hoops", self.D_hoopsD_hoops],
            ["alpha_i", self.alpha_ialpha_i],
            ["GJ", self.GJGJ],
            ["conf_mat_ID", self.conf_mat_IDconf_mat_ID],
            ["discr_core", self.discr_corediscr_core],
            ["unconf_mat_ID", self.unconf_mat_IDunconf_mat_ID],
```

```
            ["discr_cover", self.discr_coverdiscr_cover],
            ["bars_mat_ID", self.bars_mat_IDbars_mat_ID],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self, plot = False, block = False):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        """
        print("")
        print("Requested info for FibersCirc, ID = {}".format(self.IDID))
        print("Section associated: {} ".format(self.section_name_tagsection_name_tag))
        print("Base b = {} mm and concrete cover e = {} mm".format(self.bb/mm_unit, self.ee/mm_unit))
        print("Radius of the confined core r_core = {} mm, radius of the bars range r_bars = {} mm and initial angle alpha_i = {} deg".format(self.r_corer_core/mm_unit, self.r_barsr_bars/mm_unit, self.alpha_ialpha_i))
        print("Confined material model ID = {}".format(self.conf_mat_IDconf_mat_ID))
        print("Unconfined material model ID = {}".format(self.unconf_mat_IDunconf_mat_ID))
        print("Bars material model ID = {}".format(self.bars_mat_IDbars_mat_ID))
        print("Discretisation in the core [number of wedges, number of rings] = {}".format(self.discr_corediscr_core))
        print("Discretisation in the lateral covers [number of wedges, number of rings] = {}".format(self.discr_coverdiscr_cover))
        print("")

        if plot:
            plot_fiber_section(self.fib_secfib_sec, matcolor=['#808080', '#D3D3D3', 'k'])

            if block:
                plt.show()


    def CreateFibers(self):
        """
        Method that initialise the fiber by calling the OpenSeesPy commands.
        """
        create_fiber_section(self.fib_secfib_sec)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


class FibersCircRCCircShape(FibersCirc):
    """
    Class that is the children of FibersCirc and combine the class RCCircShape (section) to retrieve the information needed.

    @param FibersCirc: Parent class.
    """
    def __init__(self, ID: int, section: RCCircShape, unconf_mat_ID: int, conf_mat_ID: int, bars_mat_ID: int,
        discr_core: list, discr_cover: list, alpha_i=0.0, GJ=0):
        """
        Constructor of the class.

        @param ID (int): Unique fiber section ID.
        @param section (RCCircShape): RCCircShape section object.
        @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined fibers.
        @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
        @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars fibers.
        @param discr_core (list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges),
            number of subdivisions (fibers) in the radial direction (number of rings) for the confined core.
        @param discr_cover (list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges),
            number of subdivisions (fibers) in the radial direction (number of rings) for the unconfined cover.
        @param alpha_i (float, optional): Angle in deg of the first vertical rebars with respect to the y axis, counterclockwise. Defaults to 0.0.
        @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.
        """
        self.sectionsection = deepcopy(section)
        super().__init__(ID, section.b, section.e, section.D_bars, section.Ay, section.n_bars, section.D_hoops, unconf_mat_ID, conf_mat_ID, bars_mat_ID,
            discr_core, discr_cover, alpha_i=alpha_i, GJ=GJ)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()

```

```
class FibersIShape(Fibers):
    """
    Class that stores funcions, material properties, geometric and mechanical parameters for a steel I shape (non double symmetric) fiber section.
    Coordinates: plotting coordinte (x, y) = fiber section coordinate (z, y) = (-x, y). For more information, see the OpenSeesPy documentation.

    @param Fibers: Parent abstract class.
    """
    def __init__(self, ID: int, d, bf_t, bf_b, tf_t, tf_b, tw, top_flange_mat_ID: int, bottom_flange_mat_ID: int, web_mat_ID: int,
        discr_top_flange: list, discr_bottom_flange: list, discr_web: list,  GJ = 0.0):
        """
        Constructor of the class.

        @param ID (int): Unique fiber section ID.
        @param d (float): Depth of the section.
        @param bf_t (float): Top flange’s width of the section
        @param bf_b (float): Bottom flange’s width of the section
        @param tf_t (float): Top flange’s thickness of the section
        @param tf_b (float): Bottom flange’s thickness of the section
        @param tw (float): Web’s thickness of the section
        @param top_flange_mat_ID (int): ID of material model that will be assigned to the top flange fibers.
        @param bottom_flange_mat_ID (int): ID of material model that will be assigned to the bottom flange fibers.
        @param web_mat_ID (int): ID of material model that will be assigned to the web fibers.
        @param discr_top_flange (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top flange.
        @param discr_bottom_flange (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the bottom flange.
        @param discr_web (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the web.
        @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: d needs to be positive.
        @exception NegativeValue: bf_t needs to be positive.
        @exception NegativeValue: bf_b needs to be positive.
        @exception NegativeValue: tf_t needs to be positive.
        @exception NegativeValue: tf_b needs to be positive.
        @exception NegativeValue: tw needs to be positive.
        @exception NegativeValue: top_flange_mat_ID needs to be a positive integer.
        @exception NegativeValue: bottom_flange_mat_ID needs to be a positive integer.
        @exception NegativeValue: web_mat_ID needs to be a positive integer.
        @exception WrongDimension: discr_top_flange has a length of 2.
        @exception WrongDimension: discr_bottom_flange has a length of 2.
        @exception WrongDimension: discr_web has a length of 2.
        @exception NegativeValue: GJ needs to be positive.
        @exception InconsistentGeometry: The sum of the flanges thickness can’t be bigger than d.
        """
        # Check
        if ID < 1: raise NegativeValue()
        if d < 0: raise NegativeValue()
        if bf_t < 0: raise NegativeValue()
        if bf_b < 0: raise NegativeValue()
        if tf_b < 0: raise NegativeValue()
        if tf_t < 0: raise NegativeValue()
        if tw < 0: raise NegativeValue()
        if top_flange_mat_ID < 1: raise NegativeValue()
        if bottom_flange_mat_ID < 1: raise NegativeValue()
        if web_mat_ID < 1: raise NegativeValue()
        if len(discr_top_flange) != 2: raise WrongDimension()
        if len(discr_bottom_flange) != 2: raise WrongDimension()
        if len(discr_web) != 2: raise WrongDimension()
        if GJ < 0: raise NegativeValue()
        if tf_t+tf_b >= d: raise InconsistentGeometry()

        # Arguments
        self.IDID = ID
        self.dd = d
        self.bf_tbf_t = bf_t
        self.bf_bbf_b = bf_b
        self.tf_ttf_t = tf_t
        self.tf_btf_b = tf_b
        self.twtw = tw
        self.top_flange_mat_IDtop_flange_mat_ID = top_flange_mat_ID
        self.bottom_flange_mat_IDbottom_flange_mat_ID = bottom_flange_mat_ID
        self.web_mat_IDweb_mat_ID = web_mat_ID
        self.discr_top_flangediscr_top_flange = copy(discr_top_flange)
        self.discr_bottom_flangediscr_bottom_flange = copy(discr_bottom_flange)
        self.discr_webdiscr_web = copy(discr_web)
        self.GJGJ = GJ

        # Initialized the parameters that are dependent from others
```

```
        self.section_name_tagsection_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit()

    def ReInit(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        # Memebers
        if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag = self.section_name_tagsection_name_tag + " (modified)"

        # Parameters
        z1 = self.twtw/2
        y1 = (self.dd - self.tf_ttf_t - self.tf_btf_b)/2

        # Create the flange top
        flange_top = [y1, -self.bf_tbf_t/2, y1+self.tf_ttf_t, self.bf_tbf_t/2]
        flange_top_cmd = ['patch', 'rect', self.top_flange_mat_IDtop_flange_mat_ID, *self.discr_top_flangediscr_top_flange, *flange_top]

        # Create the flange bottom
        flange_bottom = [-y1-self.tf_btf_b, -self.bf_bbf_b/2, -y1, self.bf_bbf_b/2]
        flange_bottom_cmd = ['patch', 'rect', self.bottom_flange_mat_IDbottom_flange_mat_ID, *self.discr_bottom_flangediscr_bottom_flange, *flange_bottom]

        # Create the web
        web = [-y1, -z1, y1, z1]
        web_cmd = ['patch', 'rect', self.web_mat_IDweb_mat_ID, *self.discr_webdiscr_web, *web]

        self.fib_secfib_sec = [['section', 'Fiber', self.IDID, '-GJ', self.GJGJ],
            flange_top_cmd, web_cmd, flange_bottom_cmd]

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    # Methods
    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "FibersIShape"], # Tag for differentiating different data
            ["ID", self.IDID],
            ["section_name_tag", self.section_name_tagsection_name_tag],
            ["d", self.dd],
            ["bf_t", self.bf_tbf_t],
            ["bf_b", self.bf_bbf_b],
            ["tf_t", self.tf_ttf_t],
            ["tf_b", self.tf_btf_b],
            ["tw", self.twtw],
            ["GJ", self.GJGJ],
            ["top_flange_mat_ID", self.top_flange_mat_IDtop_flange_mat_ID],
            ["bottom_flange_mat_ID", self.bottom_flange_mat_IDbottom_flange_mat_ID],
            ["web_mat_ID", self.web_mat_IDweb_mat_ID],
            ["discr_top_flange", self.discr_top_flangediscr_top_flange],
            ["discr_bottom_flange", self.discr_bottom_flangediscr_bottom_flange],
            ["discr_web", self.discr_webdiscr_web],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self, plot = False, block = False):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the fiber. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        """
        print("")
        print("Requested info for FibersRect, ID = {}".format(self.IDID))
        print("Section associated: {} ".format(self.section_name_tagsection_name_tag))
        print("Depth d = {} mm and web thickness tw = {} mm".format(self.dd/mm_unit, self.twtw/mm_unit))
        print("Top flange width bf_t = {} mm and thickness tf_t = {} mm".format(self.bf_tbf_t/mm_unit, self.tf_ttf_t/mm_unit))
        print("Bottom flange width bf_b = {} mm and thickness tf_b = {} mm".format(self.bf_bbf_b/mm_unit, self.tf_btf_b/mm_unit))
        print("Web material model ID = {}".format(self.web_mat_IDweb_mat_ID))
        print("Top flange material model ID = {}".format(self.top_flange_mat_IDtop_flange_mat_ID))
        print("Bottom flange material model ID = {}".format(self.bottom_flange_mat_IDbottom_flange_mat_ID))
        print("Discretisation in the web [IJ or x/z dir, JK or y dir] = {}".format(self.discr_webdiscr_web))
```

```
        print("Discretisation in the top flange [IJ or x/z dir, JK or y dir] = {}".format(self.discr_top_flangediscr_top_flange))
        print("Discretisation in the bottom flange [IJ or x/z dir, JK or y dir] = {}".format(self.discr_bottom_flangediscr_bottom_flange))
        print("")

        if plot:
            plot_fiber_section(self.fib_secfib_sec, matcolor=['r', 'b', 'g', 'k'])

            if block:
                plt.show()


    def CreateFibers(self):
        """
        Method that initialise the fiber by calling the OpenSeesPy commands.
        """
        create_fiber_section(self.fib_secfib_sec)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


class FibersIShapeSteelIShape(FibersIShape):
    """
    Class that is the children of FibersIShape and combine the class SteelIShape (section) to retrieve the information needed.

    @param FibersIShape: Parent class.
    """
    def __init__(self, ID: int, section: SteelIShape, top_flange_mat_ID: int, discr_top_flange: list, discr_bottom_flange: list, discr_web: list,
        GJ=0.0, bottom_flange_mat_ID = -1, web_mat_ID = -1):
        """
        Constructor of the class.

        @param ID (int): Unique fiber section ID.
        @param section (SteelIShape): SteelIShape section object.
        @param top_flange_mat_ID (int): ID of material model that will be assigned to the top flange fibers.
        @param discr_top_flange (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top flange.
        @param discr_bottom_flange (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the bottom flange.
        @param discr_web (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the web.
        @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.
        @param bottom_flange_mat_ID (int): ID of material model that will be assigned to the bottom flange fibers.
            Defaults to -1, e.g. equal to top_flange_mat_ID.
        @param web_mat_ID (int): ID of material model that will be assigned to the web fibers.
            Defaults to -1, e.g. equal to top_flange_mat_ID.
        """
        self.sectionsection = deepcopy(section)
        if bottom_flange_mat_ID == -1: bottom_flange_mat_ID = top_flange_mat_ID
        if web_mat_ID == -1: web_mat_ID = top_flange_mat_ID

        super().__init__(ID, section.d, section.bf, section.bf, section.tf, section.tf, section.tw, top_flange_mat_ID, bottom_flange_mat_ID, web_mat_ID,
            discr_top_flange, discr_bottom_flange, discr_web, GJ)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()


def plot_fiber_section(fiber_info, fill_shapes = True, matcolor=['#808080', '#D3D3D3', 'r', 'b', 'g', 'y']):
    """
    Plot fiber cross-section. Coordinate system used: plotting coordinte = (x, y), fiber section coordinate (z, y) = (-x, y)
    Inspired by plot_fiber_section from ops_vis written by Seweryn Kokot.

    @param fiber_info (list): List of lists (be careful with the local coordinate system!). The first list defines the fiber section: \n
        ['section', 'Fiber', ID, '-GJ', GJ] \n
        The other lists have one of the following format (coordinate input: (y, z)!): \n
        ['layer', 'bar', mat_ID, A, y, z] # one bar \n
        ['layer', 'straight', mat_ID, n_bars, A, yI, zI, yJ, zJ] # line range of bars (with I = first bar, J = last bar) \n
        ['layer', 'circ', mat_ID, n_bars, A, yC, zC, r, (a0_deg), (a1_deg)] # circular range of bars (with C = center, r = radius) \n
        ['patch', 'rect', mat_ID, *discr, -yI, zI, yK, -zK] # rectangle (with yI = yK = d/2; zI = zK = b/2) \n
        ['patch', 'quad', mat_ID, *discr, yI, zI, yJ, zJ, yK, zK, yL, zL] # quadrilateral shaped (starting from bottom left, counterclockwise: I, J, K, L) \n
        ['patch', 'circ', mat_ID, *discr, yC, zC, ri, re, (a0), (a1)] #  (with C = center, ri = internal radius, re = external radius)
    @param fill_shapes (bool, optional): Option to fill fibers with color specified in matcolor.
```

```
Defaults to True.
    @param matcolor (list, optional): List of colors for various material IDs. Defaults to ['#808080', '#D3D3D3', 'r', 'b', 'g', 'y'].

    Example 1: Simple rectangle with 2 rebars (D = diameter) on top (e distance from the top and from the lateral borders).
        Rectangle with first corner =  I (bottom right) and second corner = K (top left); number of fibers = discr (list of 2)
        fib_sec = [['section', 'Fiber', ID, '-GJ', GJ],
            ['patch', 'rect', concrete_mat_ID, *discr, yI, zI, yK, zK],
            ['layer', 'bar', bars_mat_ID, Ay, yI-e-D/2, zI-e-D/2], # left rebar
            ['layer', 'bar', bars_mat_ID, Ay, yI-e-D/2, -(zI-e-D/2)]] # right rebar

    Example 2: double symmetric I shape.
        Each rectangle (2 flanges and 1 web): first corner =  I (bottom right) and second corner = K (top left); number of fibers = discr (list of 2)
        fib_sec = [['section', 'Fiber', ID, '-GJ', GJ],
            ['patch', 'rect', mat_ID, *discr, yI_tf, zI_tf, yK_tf, zK_tf], # top flange
            ['patch', 'rect', mat_ID, *discr, yI_bf, zI_bf, yK_bf, zK_bf], # bottom flange
            ['patch', 'rect', mat_ID, *discr, yI_w, zI_w, yK_w, zK_w]] # web
    """

    mat_to_col = {}
    fig, ax = plt.subplots()
    ax.grid(False)

    for item in fiber_info:
        if item[0] == 'section':
            fib_ID = item[2]
        if item[0] == 'layer':
            matID = item[2]
            mat_to_col = __assignColorToMat(matID, mat_to_col, matcolor)
            if item[1] == 'bar':
                As = item[3]
                Iy = item[4]
                Iz = item[5]
                r = np.sqrt(As / np.pi)
                bar = Circle((-Iz, Iy), r, ec='k', fc='k', zorder=10)
                ax.add_patch(bar)
            if item[1] == 'straight':
                n_bars = item[3]
                As = item[4]
                Iy, Iz, Jy, Jz = item[5], item[6], item[7], item[8]
                r = np.sqrt(As / np.pi)
                Y = np.linspace(Iy, Jy, n_bars)
                Z = np.linspace(Iz, Jz, n_bars)
                for zi, yi in zip(Z, Y):
                    bar = Circle((-zi, yi), r, ec='k', fc=mat_to_col[matID], zorder=10)
                    ax.add_patch(bar)
            if item[1] == 'circ':
                n_bars, As = item[3], item[4]
                yC, zC, r = item[5], item[6], item[7]
                if len(item) > 8:
                    a0_deg = item[8]
                else:
                    a0_deg = 0.
                a1_deg = 360. - 360./n_bars + a0_deg
                if len(item) > 9: a1_deg = item[9]

                a0_rad, a1_rad = np.pi * a0_deg / 180., np.pi * a1_deg / 180.
                r_bar = np.sqrt(As / np.pi)
                thetas = np.linspace(a0_rad, a1_rad, n_bars)
                Y = yC + r * np.cos(thetas)
                Z = zC + r * np.sin(thetas)
                for zi, yi in zip(Z, Y):
                    bar = Circle((-zi, yi), r_bar, ec='k', fc=mat_to_col[matID], zorder=10)
                    ax.add_patch(bar)

        if (item[0] == 'patch' and (item[1] == 'quad' or item[1] == 'quadr' or
                                    item[1] == 'rect')):
            matID, nIJ, nJK = item[2], item[3], item[4]
            mat_to_col = __assignColorToMat(matID, mat_to_col, matcolor)


            if item[1] == 'quad' or item[1] == 'quadr':
                Iy, Iz, Jy, Jz = item[5], item[6], item[7], item[8]
                Ky, Kz, Ly, Lz = item[9], item[10], item[11], item[12]

            if item[1] == 'rect':
                Iy, Iz, Ky, Kz = item[5], item[6], item[7], item[8]
                Jy, Jz, Ly, Lz = Iy, Kz, Ky, Iz
                # check order of definition
                if Kz-Iz < 0 or Ky-Iy < 0: print("!!!!!!! WARNING !!!!!!! The fiber is not defined bottom right, top left")

            # check for convexity (vector products)
            outIJxIK = (Jy-Iy)*(Kz-Iz) - (Ky-Iy)*(Jz-Iz)
```

```
            outIKxIL = (Ky-Iy)*(Lz-Iz) - (Ly-Iy)*(Kz-Iz)
            # check if I, J, L points are colinear
            outIJxIL = (Jy-Iy)*(Lz-Iz) - (Ly-Iy)*(Jz-Iz)
            # outJKxJL = (Ky-Jy)*(Lz-Jz) - (Ly-Jy)*(Kz-Jz)

            if -outIJxIK <= 0 or -outIKxIL <= 0 or -outIJxIL <= 0:
                print('!!!!!!! WARNING !!!!!!! Patch quad is non-convex or non-counter-clockwise
defined or has at least 3 colinear points in line')

            IJz, IJy = np.linspace(Iz, Jz, nIJ+1), np.linspace(Iy, Jy, nIJ+1)
            JKz, JKy = np.linspace(Jz, Kz, nJK+1), np.linspace(Jy, Ky, nJK+1)
            LKz, LKy = np.linspace(Lz, Kz, nIJ+1), np.linspace(Ly, Ky, nIJ+1)
            ILz, ILy = np.linspace(Iz, Lz, nJK+1), np.linspace(Iy, Ly, nJK+1)

            if fill_shapes:
                Z = np.zeros((nIJ+1, nJK+1))
                Y = np.zeros((nIJ+1, nJK+1))

                for j in range(nIJ+1):
                    Z[j, :] = np.linspace(IJz[j], LKz[j], nJK+1)
                    Y[j, :] = np.linspace(IJy[j], LKy[j], nJK+1)

                for j in range(nIJ):
                    for k in range(nJK):
                        zy = np.array([[-Z[j, k], Y[j, k]],
                                       [-Z[j, k+1], Y[j, k+1]],
                                       [-Z[j+1, k+1], Y[j+1, k+1]],
                                       [-Z[j+1, k], Y[j+1, k]]])
                        poly = Polygon(zy, True, ec='k', fc=mat_to_col[matID])
                        ax.add_patch(poly)

            else:
                # horizontal lines
                for az, bz, ay, by in zip(IJz, LKz, IJy, LKy):
                    plt.plot([-az, -bz], [ay, by], 'b-', zorder=1)

                # vertical lines
                for az, bz, ay, by in zip(JKz, ILz, JKy, ILy):
                    plt.plot([-az, -bz], [ay, by], 'b-', zorder=1)

        if item[0] == 'patch' and item[1] == 'circ':
            matID, nc, nr = item[2], item[3], item[4]
            mat_to_col = __assignColorToMat(matID, mat_to_col, matcolor)

            yC, zC, ri, re = item[5], item[6], item[7], item[8]
            if len(item) > 9:
                a0 = item[9]
            else:
                a0 = 0.
            a1 = 360. + a0
            if len(item) > 10: a1 = item[10]

            dr = (re - ri) / nr
            dth = (a1 - a0) / nc

            for j in range(nr):
                rj = ri + j * dr
                rj1 = rj + dr

                for i in range(nc):
                    thi = a0 + i * dth
                    thi1 = thi + dth
                    wedge = Wedge((-zC, yC), rj1, thi, thi1, width=dr, ec='k', #Seweryn Kokot: (yC,
-zC), wrong??
                        lw=1, fc=mat_to_col[matID])
                    ax.add_patch(wedge)
    ax.set(xlabel='x dimension [{}]'.format(length_unit), ylabel='y dimension
[{}]'.format(length_unit),
                    title='Fiber section (ID = {})'.format(fib_ID))
    ax.axis('equal')


def __assignColorToMat(matID: int, mat_to_col: dict, matcolor: list):
    """
    PRIVATE FUNCTION. Used to assign different colors for each material model assign to the fiber
section.

    @param matID (int): ID of the material model.
    @param mat_to_col (dict): Dictionary to check with material model has which color.
    @param matcolor (list): List of colors.

    @returns dict: Updated dictionary.
    """
    if not matID in mat_to_col:
        if len(mat_to_col) >= len(matcolor):
            print("Warning: not enough colors defined for fiber section plot (white used)")
            mat_to_col[matID] = 'w'
```

```
        else:
            mat_to_col[matID] = matcolor[len(mat_to_col)]
    return mat_to_col


def create_fiber_section(fiber_info):
    """
    Initialise fiber cross-section with OpenSeesPy commands.
    For examples, see plot_fiber_section.
    Inspired by fib_sec_list_to_cmds from ops_vis written by Seweryn Kokot

    @param fiber_info (list): List of lists (be careful with the local coordinate system!). The first list defines the fiber section: \n
        ['section', 'Fiber', ID, '-GJ', GJ] \n
        The other lists have one of the following format (coordinate input: (y, z)!): \n
        ['layer', 'bar', mat_ID, A, y, z] # one bar \n
        ['layer', 'straight', mat_ID, n_bars, A, yI, zI, yJ, zJ] # line range of bars (with I = first bar, J = last bar) \n
        ['layer', 'circ', mat_ID, n_bars, A, yC, zC, r, (a0_deg), (a1_deg)] # circular range of bars (with C = center, r = radius) \n
        ['patch', 'rect', mat_ID, *discr, -yI, zI, yK, -zK] # rectangle (with yI = yK = d/2; zI = zK = b/2) \n
        ['patch', 'quad', mat_ID, *discr, yI, zI, yJ, zJ, yK, zK, yL, zL] # quadrilateral shaped (starting from bottom left, counterclockwise: I, J, K, L) \n
        ['patch', 'circ', mat_ID, *discr, yC, zC, ri, re, (a0), (a1)] #  (with C = center, ri = internal radius, re = external radius)
    """
    for dat in fiber_info:
        if dat[0] == 'section':
            fib_ID, GJ = dat[2], dat[4]
            section('Fiber', fib_ID, 'GJ', GJ)

        if dat[0] == 'layer':
            mat_ID = dat[2]
            if dat[1] == 'straight':
                n_bars = dat[3]
                As = dat[4]
                Iy, Iz, Jy, Jz = dat[5], dat[6], dat[7], dat[8]
                layer('straight', mat_ID, n_bars, As, Iy, Iz, Jy, Jz)
            if dat[1] == 'bar':
                As = dat[3]
                Iy = dat[4]
                Iz = dat[5]
                fiber(Iy, Iz, As, mat_ID)
                # layer('straight', mat_ID, 1, As, Iy, Iz, Iy, Iz)
            if dat[1] == 'circ':
                n_bars, As = dat[3], dat[4]
                yC, zC, r = dat[5], dat[6], dat[7]
                if len(dat) > 8:
                    a0_deg = dat[8]
                else:
                    a0_deg = 0.
                a1_deg = 360. - 360./n_bars + a0_deg
                if len(dat) > 9: a1_deg = dat[9]
                layer('circ', mat_ID, n_bars, As, yC, zC, r, a0_deg, a1_deg)

        if dat[0] == 'patch':
            mat_ID = dat[2]
            nIJ = dat[4]
            nJK = dat[3]

            if dat[1] == 'quad' or dat[1] == 'quadr':
                Iy, Iz, Jy, Jz = dat[5], dat[6], dat[7], dat[8]
                Ky, Kz, Ly, Lz = dat[9], dat[10], dat[11], dat[12]
                patch('quad', mat_ID, nIJ, nJK, Iy, Iz, Jy, Jz, Ky, Kz,
                        Ly, Lz)

            if dat[1] == 'rect':
                Iy, Iz, Ky, Kz = dat[5], dat[6], dat[7], dat[8]
                patch('rect', mat_ID, nIJ, nJK, Iy, Iz, Ky, Kz)
                # patch('rect', mat_ID, nIJ, nJK, Iy, Kz, Ky, Iz)

            if dat[1] == 'circ':
                mat_ID, nc, nr = dat[2], dat[3], dat[4]
                yC, zC, ri, re = dat[5], dat[6], dat[7], dat[8]
                if len(dat) > 9:
                    a0 = dat[9]
                else:
                    a0 = 0.
                a1 = 360. + a0
                if len(dat) > 10: a1 = dat[10]
                patch('circ', mat_ID, nc, nr, yC, zC, ri, re, a0, a1)

```

8.13 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/↵ FunctionalFeatures.py File Reference

Classes

- class IDGenerator

  Class that manage the ID generation.

Namespaces

- namespace FunctionalFeatures

  Module with useful functions (discretise curves, ID conventions, etc)
  Carmine Schipani, 2021.

Functions

- def DiscretizeLinearly (np.ndarray LP, int discr, plot=False, block=False, show_original_LP=True)

  This function discretize the curve 'LP' given adding the number of point given by 'discr' between every point (linearly).
- def DiscretizeLoadProtocol (np.ndarray SDR_LP, np.ndarray nr_cycles_LP, int discr_first_cycle, plot=False, block=False, show_original_peaks=True)

  Discretized a cyclic load protocol keeping a similar discretisation step throughout the different cycles and keeping in the output the extremes (peaks).
- def GridIDConvention (int pier_axis, int floor_axis, max_pier=-1, max_floor=-1)

  Function used to construct the ID of the nodes in the grid (first nodes that define the geometry of the model).
- def IDConvention (int prefix, int suffix, n_zeros_between=0)

  Function used to construct IDs for elements and offgrid nodes.
- def NodesOrientation (int iNode_ID, int jNode_ID)

  Function that finds the orientation of the vector with direction 'jNode_ID''iNode_ID'.
- def OffsetNodeIDConvention (int node_ID, str orientation, str position_i_or_j)

  Function used to add node on top of existing ones in the extremes of memebers with springs.
- def plot_member (list element_array, member_name="Member name not defined", show_element_ID=True, show_node_ID=True)

  Function that plots a set of elements.
- def plot_nodes (list nodes_array, name="Not defined", show_node_ID=True)

  Function that plots a set of nodes.
- def ProgressingPercentage (max_iter, int i, int next_step, step=10)

  Function that shows the progressing percentage of an iterative process.

8.14 FunctionalFeatures.py

Go to the documentation of this file.

```
"""Module with useful functions (discretise curves, ID conventions, etc) \n
Carmine Schipani, 2021
"""

import math
import numpy as np
import matplotlib.pyplot as plt
import openseespy.postprocessing.internal_plotting_functions as ipltf
from openseespy.opensees import *
from OpenSeesPyAssistant.ErrorHandling import *
from OpenSeesPyAssistant.Units import *
```

```
def ProgressingPercentage(max_iter, i: int, next_step: int, step = 10):
    """
    Function that shows the progressing percentage of an iterative process.

    @param max_iter (int): Maximal number of interations
    @param i (int): Current iteration
    @param next_step (int): Next step of the percentage (set to 0 for the first iteration and then use the return parameter)
    @param step (int, optional): Size of the step (should be a fraction of 100). Defaults to 10.

    @returns int: The updated next step
    """
    if i*100.0/(max_iter-1) >= next_step:
        print("The progression is {}%".format(next_step))
        return next_step + step

    return next_step


def DiscretizeLoadProtocol(SDR_LP: np.ndarray, nr_cycles_LP: np.ndarray, discr_first_cycle: int, plot = False, block = False, show_original_peaks = True):
    """
    Discretized a cyclic load protocol keeping a similar discretisation step throughout the different cycles and keeping in the output the extremes (peaks).

    @param SDR_LP (np.ndarray): Array (1 dimension) that stores the peaks of the cycles.
        They needs to be only the positive peaks, beacuse this function will use them as the extreme for each cycle.
    @param nr_cycles_LP (np.ndarray): Array (1 dimension) that stores the number of cycles for every extreme declared in 'SDR_LP' and its countepart negative.
        They need to be positive integers.
    @param discr_first_cycle (int): The number of points from peak to peak (counting the two peaks) in the first cycle. It should be odd.
    @param plot (bool, optional): Option to show the plot of the discretized (and also the original peaks). Defaults to False.
    @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
    @param show_original_peaks (bool, optional): Option to show the original peaks to check if the discretized curve is correct.
        The argument plot need to be True. Defaults to True.

    @exception WrongDimension: SDR_LP and nr_cycles_LP need to be of same length.
    @exception NegativeValue: SDR_LP needs to have only positive integers.
    @exception NegativeValue: nr_cycles_LP needs to have only positive integers.
    @exception NegativeValue: discr_first_cycle needs to be a positive integer.

    @returns np.array: Array (1 dimension) that stores the new discretized load protocol curve.
    """
    if np.size(SDR_LP) != np.size(nr_cycles_LP): raise WrongDimension()
    if any(col < 0 for col in SDR_LP): raise NegativeValue()
    if any(col < 0 for col in nr_cycles_LP): raise NegativeValue()
    if discr_first_cycle < 0: raise NegativeValue()

    if discr_first_cycle % 2 == 0:
            discr_first_cycle = discr_first_cycle + 1
    discr_factor = discr_first_cycle / (SDR_LP[0]*2)
    discretized_LP = [0.0]
    x_val = []
    skip_x = 0
    for i in range(np.size(SDR_LP)):
        discr_i = math.ceil(discr_factor*SDR_LP[i]*2)-1;
        if discr_i % 2 == 0:
            discr_i = discr_i + 1
        length_tmp = int((discr_i+1)/2)
        tmp_up = np.linspace(0.0, SDR_LP[i], length_tmp)
        tmp_down = np.linspace(SDR_LP[i], 0.0, length_tmp)
        for j in range(int(nr_cycles_LP[i])):
            discretized_LP = np.append(discretized_LP, tmp_up[1:length_tmp])
            discretized_LP = np.append(discretized_LP, tmp_down[1:length_tmp])
            discretized_LP = np.append(discretized_LP, -tmp_up[1:length_tmp])
            discretized_LP = np.append(discretized_LP, -tmp_down[1:length_tmp])
        # for check of original peaks
        x_val.append(length_tmp-1+skip_x)
        skip_x = (length_tmp-1)*(4*(nr_cycles_LP[i]-1)+3)+x_val[-1]


    if plot:
        fig, ax = plt.subplots()
        ax.plot(discretized_LP, '-r', label='Discretised LP')

        ax.set(xlabel='Step number [-]', ylabel='Unit of the loading protocol',
            title='Discretized loading protocol')
        ax.grid()

        if show_original_peaks:
```

```
            ax.plot(x_val, SDR_LP, 'ob', label='Original LP')
            ax.legend()

        if block:
            plt.show()

    return discretized_LP


def DiscretizeLinearly(LP: np.ndarray, discr: int, plot = False, block = False, show_original_LP = True):
    """
    This function discretize the curve 'LP' given adding the number of point given by 'discr' between every point (linearly).

    @param LP (np.ndarray): Array (1 dimension) that stores the curve that needs to be discretized
    @param discr (int): The number of points to add between every two points of 'LP' (linearly)
    @param plot (bool, optional): Option to show the plot of the discretized (and also the original LP). Defaults to False.
    @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
    @param show_original_LP (bool, optional): Option to show the original LP to check if the discretized curve is correct. Defaults to True.

    @returns np.ndarray: Array (1 dimension) that stores the new discretized load protocol.
    """

    #TODO: check discr nonnegative int and LP 1 dimension

    # Define the new discretized LP
    length = 1 + (np.size(LP)-1) * (discr+1)
    discr_LP = np.zeros(length)

    # Performa manually the first iteration
    yprev = LP[0]
    x = [0, 1]
    discr_LP[0] = yprev
    iter = 0

    # add the other points and the discretized ones
    for ynext in LP[1:]:
        y = [yprev, ynext]

        # Compute new points
        xnew = np.linspace(x[0], x[1], discr+2)
        ynew = np.interp(xnew[1:], x, y)

        # Add to the recording vector discr_LP
        index = np.array(np.arange(discr+1)+1+iter)
        discr_LP[index] = ynew

        # Prepare for next iteration
        yprev = ynext
        iter = iter + discr + 1

    if plot:
        fig, ax = plt.subplots()
        ax.plot(discr_LP, '-r', label='Discretised LP')

        ax.set(xlabel='Step number [-]', ylabel='Unit of the loading protocol',
            title='Discretized loading protocol')
        ax.grid()

        if show_original_LP:
            x_val = np.arange(0, np.size(discr_LP), discr+1)
            ax.plot(x_val, LP, 'ob', label='Original LP')
            ax.legend()

        if block:
            plt.show()

    return discr_LP


def GridIDConvention(pier_axis: int, floor_axis: int, max_pier = -1, max_floor = -1):
    """
    Function used to construct the ID of the nodes in the grid (first nodes that define the geometry of the model).
    The conventional grid node ID is xy, with x = the pier position 'pier_axis'; y = the floor position 'floor_axis'.

    @param pier_axis (int): The pier (or x) postion of the node.
    @param floor_axis (int): The floor (or y) position of the node.
    @param max_pier (int, optional): Maximal pier position of the model (used to identify the number of digits).
        Defaults to -1, e.g. taken equal of pier_axis.
    @param max_floor (int, optional): Maximal floor position of the model (used to identify the number
```

```
of digits).
        Defaults to -1, e.g. taken equal of floor_axis.

    @exception NameError: Work In Progress: only 9 floors/bays.
    @exception NegativeValue: The argument pier_axis needs to be a positive integer.
    @exception NegativeValue: The argument floor_axis needs to be a positive integer.
    @exception NegativeValue: The argument max_pier needs to be a positive integer if different from -1.
    @exception NegativeValue: The argument max_floor needs to be a positive integer if different from -1.
    @exception WrongArgument: The argument max_pier need to be equal or bigger to pier_axis
    @exception WrongArgument: The argument max_floor need to be equal or bigger to floor_axis

    @returns int: The grid node ID
    """
    # Convention:
    #       GridNodeID:     xy          with x = pier, y = floor
    if pier_axis > 9 or floor_axis > 9 or max_pier > 9 or max_floor > 9: raise NameError("WIP: maximal 9 floors or bays")
    max_pier = pier_axis if max_pier == -1 else max_pier
    max_floor = floor_axis if max_floor == -1 else max_floor

    if pier_axis < 0: raise NegativeValue()
    if floor_axis < 0: raise NegativeValue()
    if max_pier < 0: raise NegativeValue()
    if max_floor < 0: raise NegativeValue()
    if max_pier < pier_axis: raise WrongArgument()
    if max_floor < floor_axis: raise WrongArgument()

    max_x_digits = int(math.log10(max_pier))+1
    max_y_digits = int(math.log10(max_floor))+1

    # return 10**(max_x_digits+max_y_digits) + pier_axis*10**max_y_digits + floor_axis # with 1 as prefix (to consider more than one digit per axis, but exceed max ID)
    return pier_axis*10**max_y_digits + floor_axis


def IDConvention(prefix: int, suffix: int, n_zeros_between = 0):
    """
    Function used to construct IDs for elements and offgrid nodes.
    It appends to a positive integer number 'prefix' a number of zeros 'n_zeros_between' and at last another positive integer 'suffix'.
    The conventional element ID is xy(a)x'y'(a') with xya = the node ID in pier x, floor y and offgrid parameter a (optional);
        x'y'a' = the node ID in pier x', floor y' and offgrid parameter a' (optional).
    For more information on x and y, see GridIDConvention; for more information on a, see OffsetNodeIDConvention.

    @param prefix (int): Prefix of the new ID. For a vertical element it should be the left node ID;
        for an horizontal one it should be the bottom node.
    @param suffix (int): Suffix of the new ID. For a vertical element it should be the right node ID;
        for an horizontal one it should be the top node.
    @param n_zeros_between (int, optional): Number of zeros to add between the two nodes. Defaults to 0.

    @exception NegativeValue: The argument prefix needs to be a positive integer.
    @exception NegativeValue: The argument suffix needs to be a positive integer.
    @exception NegativeValue: The argument n_zeros_between needs to be a positive integer.

    @returns int: The combined ID
    """
    # Convention:
    #       ElementID:      xy(a)x'y'(a')   with xy(a) = NodeID i and x'y'(a') = NodeID j
    #       TrussID:        xy(a)x'y'(a')   with xy(a) = NodeID i and x'y'(a') = NodeID j
    #       Spring:         xy(a)x'y'(a')   with xy(a) = NodeID i and x'y'(a') = NodeID j
    if prefix < 0: raise NegativeValue()
    if suffix < 0: raise NegativeValue()
    if n_zeros_between < 0: raise NegativeValue()

    return int(str(prefix*10**n_zeros_between) + str(suffix))


def OffsetNodeIDConvention(node_ID: int, orientation: str, position_i_or_j: str):
    """
    Function used to add node on top of existing ones in the extremes of memebers with springs.

    @param node_ID (int): Node that we refer to.
    @param orientation (str): Orientation of the memeber. Can be 'vertical' or 'horizontal'.
    @param position_i_or_j (str): Position at the start 'i' (left or bottom)
        or at the end 'j' (right or top) of 'node_ID' in the member.

    @exception NegativeValue: The argument node_ID needs to be a positive integer.
    @exception WrongArgument: The argument position_i_or_j needs to be 'i' or 'j'
    @exception WrongArgument: The argument orientation needs to be 'vertical' or 'horizontal'

    @returns int: The combined ID
    """
```

```
    # Convention:
      o xy
    #       GridNodeID:      xy             with x = pier, y = floor
      o xy7
    #   AdditionalNodeID:    xya            with x = pier, y = floor, a:  o xy  xy2 o-------o x'y3
  x'y o   |
    #   PanelZoneNodeID:     xy(a)a     see MemberModel for the panel zone ID convention
      |
    #
      |
    #
      o xy'6
    #
      o xy'
    if node_ID < 1: raise NegativeValue()
    if position_i_or_j != "i" and position_i_or_j != "j": raise WrongArgument()

    if orientation == "vertical":
        if position_i_or_j == "i":
            return IDConvention(node_ID, 6)
        else:
            return IDConvention(node_ID, 7)
    elif orientation == "horizontal":
        if position_i_or_j == "i":
            return IDConvention(node_ID, 2)
        else:
            return IDConvention(node_ID, 3)
    else: raise WrongArgument()


def NodesOrientation(iNode_ID: int, jNode_ID: int):
    """
    Function that finds the orientation of the vector with direction 'jNode_ID"iNode_ID'.
    If the the nodes are on top of each other, the function returns 'zero_length'.

    @param iNode_ID (int): Node i.
    @param jNode_ID (int): Node j.

    @exception NegativeValue: The argument iNode_ID needs to be a positive integer.
    @exception NegativeValue: The argument jNode_ID needs to be a positive integer.

    @returns str: The orientation of the vector.
    """
    if iNode_ID < 1: raise NegativeValue()
    if jNode_ID < 1: raise NegativeValue()

    iNode = np.array(nodeCoord(iNode_ID))
    jNode = np.array(nodeCoord(jNode_ID))
    if abs(iNode[0]-jNode[0]) + abs(iNode[1]-jNode[1]) == 0:
        return "zero_length"
    elif abs(iNode[0]-jNode[0]) < abs(iNode[1]-jNode[1]):
        return "vertical"
    else:
        return "horizontal"


def plot_member(element_array: list, member_name = "Member name not defined", show_element_ID = True,
  show_node_ID = True):
    """
    Function that plots a set of elements. It can be used to check the correctness of a part of the
  model or of a member.
    If the entire model need to be plotted, use instead 'plot_model("nodes", "elements")' from
  openseespy.postprocessing.Get_Rendering. \n
    Inspired by plot_model written by Anurag Upadhyay and Christian Slotboom.

    @param element_array (list): An array (list of lists of one dimensions and length = 3) that store
  the element and nodes IDs.
        An element is stored in one list with 3 entries: the element ID, node i ID and node j ID.
    @param member_name (str, optional): The name of what is plotted. Defaults to "Member name not
  defined".
    @param show_element_ID (bool, optional): Option to show the element IDs. Defaults to True.
    @param show_node_ID (bool, optional): Option to show the nodes IDs. Defaults to True.

    @exception WrongDimension: element_array needs to be non-empty.
    @exception WrongDimension: The number of entries in the lists inside the argument element_array
  need to be 3.

    @returns matplotlib.axes._subplots.AxesSubplo: The figure's wrappr, useful to customise the plot
  (change axes label, etc).
    """
    if len(element_array) == 0: raise WrongArgument()
    if len(element_array[0]) != 3: raise WrongDimension()

    node_style = {'color':'black', 'marker':'o', 'facecolor':'black','linewidth':0.}
    node_text_style = {'fontsize':8, 'fontweight':'regular', 'color':'green'}
    track_node = {}

```

```
    if show_element_ID:
        show_e_ID = 'yes'
    else:
        show_e_ID = 'no'

    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)

    for ele in element_array:
        eleTag = int(ele[0])
        Nodes =ele[1:]

        if len(Nodes) == 2:
            # 2D element
            iNode = np.array(nodeCoord(Nodes[0]))
            jNode = np.array(nodeCoord(Nodes[1]))
            ipltf._plotBeam2D(iNode, jNode, ax, show_e_ID, eleTag, "solid")
            ax.scatter(*iNode, **node_style)
            ax.scatter(*jNode, **node_style)
            if show_node_ID:
                if abs(sum(iNode - jNode)) > 1e-6:
                    # beam-col
                    __plt_node(Nodes[0], track_node, iNode, ax, node_text_style)
                    __plt_node(Nodes[1], track_node, jNode, ax, node_text_style, h_align='right', v_align='bottom')
                else:
                    # zerolength
                    __plt_node(Nodes[0], track_node, iNode, ax, node_text_style, h_align='right')
                    __plt_node(Nodes[1], track_node, jNode, ax, node_text_style)
        else:
            print("Too many nodes in this elemnet (see shell elements)")

    ax.set_xlabel('x [{}]'.format(length_unit))
    ax.set_ylabel('y [{}]'.format(length_unit))
    plt.title("Visualisation of: {}".format(member_name))
    plt.axis('equal')
    return ax


def plot_nodes(nodes_array: list, name = "Not defined", show_node_ID = True):
    """
    Function that plots a set of nodes. It can be used to check the correctness of the model's geometry.
    If the entire model need to be plotted, use instead 'plot_model("nodes", "elements")' from openseespy.postprocessing.Get_Rendering.

    @param nodes_array (list): List of 1 dimension with the IDs of the nodes to be displayed.
    @param name (str, optional): Name that describe what the plot will show. Defaults to "Not defined".
    @param show_node_ID (bool, optional): Option to show the node IDs. Defaults to True.

    @exception WrongArgument: nodes_array needs to be non-empty.

    @returns (matplotlib.axes._subplots.AxesSubplot): The figure's wrapper, useful to customise the plot (change axes label, etc).
    """
    if len(nodes_array) == 0: raise WrongArgument()

    node_style = {'color':'black', 'marker':'o', 'facecolor':'black','linewidth':0.}
    node_text_style = {'fontsize':8, 'fontweight':'regular', 'color':'green'}
    track_node = {}

    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)

    for node_ID in nodes_array:
        node_xy = np.array(nodeCoord(node_ID))
        ax.scatter(*node_xy, **node_style)
        if show_node_ID:
            __plt_node(node_ID, track_node, node_xy, ax, node_text_style)

    ax.set_xlabel('x [{}]'.format(length_unit))
    ax.set_ylabel('y [{}]'.format(length_unit))
    plt.title("Visualisation of: {}".format(name))
    plt.axis('equal')
    return ax


def __plt_node(nodeID: int, track_node: dict, NodeXY, ax, node_text_style, x_off = 0, y_off = 0, h_align = 'left', v_align='top'):
    """
    PRIVATE FUNCTION. Used to plot the nodes in a controlled way (no repetition, position of the IDs, text style).

    @param nodeID (int): The ID of the node.
    @param track_node (dict): A dictionary used to avoid plotting a node multiple times.
    @param NodeXY (list): List of dimension 1, length 2 with the position of the node.
```

```
    @param ax (matplotlib.axes._subplots.AxesSubplot): The figure's wrappr.
    @param node_text_style (dict): Dictionary for the text style.
    @param x_off (int, optional): Offset in x direction. Defaults to 0.
    @param y_off (int, optional): Offset in y direction. Defaults to 0.
    @param h_align (str, optional): Horizontal alignment ('left' or 'right'). Defaults to 'left'.
    @param v_align (str, optional): Vertical alignment ('center', 'top' or 'bottom'). Defaults to 'top'.
    """
    if not nodeID in track_node:
        track_node[nodeID] = True
        ax.text(NodeXY[0]+x_off, NodeXY[1]+y_off, nodeID,**node_text_style, horizontalalignment=h_align, verticalalignment=v_align)


class IDGenerator():
    """Class that manage the ID generation.
    USE ONLY IF EVERY NODE IS DEFINED BY THE USER (because the OpenSeesPyAssistant modules use the convention defined in the functions above).
    """
    def __init__(self):
        """The class constructor.
        """
        self.current_node_IDcurrent_node_ID = 0
        self.current_element_IDcurrent_element_ID = 0
        self.current_mat_IDcurrent_mat_ID = 0
        self.current_fiber_IDcurrent_fiber_ID = 0

    def GenerateIDNode(self):
        """
        Method that generate a unique node ID.

        @returns int: The node ID.
        """
        self.current_node_IDcurrent_node_ID = self.current_node_IDcurrent_node_ID + 1
        return self.current_node_IDcurrent_node_ID

    def GenerateIDElement(self):
        """
        Method that generate a unique element ID.

        @returns int: The element ID.
        """
        self.current_element_IDcurrent_element_ID = self.current_element_IDcurrent_element_ID + 1
        return self.current_element_IDcurrent_element_ID

    def GenerateIDMat(self):
        """
        Method that generate a unique material ID.

        @returns int: The material ID.
        """
        self.current_mat_IDcurrent_mat_ID = self.current_mat_IDcurrent_mat_ID + 1
        return self.current_mat_IDcurrent_mat_ID

    def GenerateIDFiber(self):
        """
        Method that generate a unique fiber ID.

        @returns int: The fiber ID.
        """
        self.current_fiber_IDcurrent_fiber_ID = self.current_fiber_IDcurrent_fiber_ID + 1
        return self.current_fiber_IDcurrent_fiber_ID

```

8.15 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/GeometryTemplate.py File Reference

Namespaces

- namespace GeometryTemplate

    Module with geometry templates (nodes and/or elements with associated fibers, material models, etc).

Functions

- def DefineFrameNodes (int n_hor_axis, int n_vert_axis, storey_width, storey_height, half_pz_height=np.array([ ]), origin=[0, 0], first_hor_axis=1, first_vert_axis=1, show_plot=True)

    Function that declares and initialises the grid nodes of a frame.
- def DefineFrameNodesAndElementsSteelIShape (int n_hor_axis, int n_vert_axis, storey_width, storey_height, list list_col, list list_beam, int geo_trans_ID, N_G=np.array([ ]), t_dp=np.array([ ]), L_b_col=np.array([ ]), L_b_beam=np.array([ ]), fix_support=True, show_plot=True, panel_zone=True)

    WIP (Work In Progress).
- def DefineSubassemblageNodes (beam_left_L_cl, beam_right_L_cl, col_top_L_cl, col_bottom_L_cl, depth_col, depth_beam, boundary_condition=True, show_plot=True)

    Function that declares and initialises the grid nodes of an interior subassemblage.
- def Initialize2DModel (data_dir="Results")

    Function that initialise the project creating the 2D model with 3 DOF per node and set up a directory for the results.

8.16 GeometryTemplate.py

Go to the documentation of this file.

```
"""
Module with geometry templates (nodes and/or elements with associated fibers, material models, etc).
Carmine Schipani, 2021
"""

# Import libraries
from openseespy.opensees import *
import matplotlib.pyplot as plt
import numpy as np
import os
from copy import copy, deepcopy
import openseespy.postprocessing.Get_Rendering as opsplt
from OpenSeesPyAssistant.Section import *
from OpenSeesPyAssistant.DataManagement import *
from OpenSeesPyAssistant.ErrorHandling import *
from OpenSeesPyAssistant.Units import *
from OpenSeesPyAssistant.Constants import *
from OpenSeesPyAssistant.Fibers import *
from OpenSeesPyAssistant.Connections import *
from OpenSeesPyAssistant.FunctionalFeatures import *
from OpenSeesPyAssistant.MemberModel import *
from OpenSeesPyAssistant.AnalysisAndPostProcessing import *


def Initialize2DModel(data_dir = "Results"):
    """
    Function that initialise the project creating the 2D model with 3 DOF per node and set up a directory for the results.

    @param data_dir (str, optional): Directory where the data will be stored.
        The function forces the user to define it just for good practice and consistency between projects.
        Defaults to "Results".
    """
    # Clear all
    wipe()

    # Build model (2D - 3 DOF/node)
    model('basic', '-ndm', 2, '-ndf', 3)

    # Main Results Folder
    if not os.path.exists(data_dir):
        os.makedirs(data_dir)


def DefineFrameNodes(n_hor_axis: int, n_vert_axis: int, storey_width, storey_height, half_pz_height = np.array([]),
    origin = [0, 0], first_hor_axis = 1, first_vert_axis = 1, show_plot = True):
    """
    Function that declares and initialises the grid nodes of a frame. Option to offset the grid node of the panel zones
        with the master node of the panel zone being the grid one (top center one). The function can be used multiple times
        to create more complex geometries.

```

```
    @param n_hor_axis (int): Number of horizontal axis (or piers) for the grid of the frame.
    @param n_vert_axis (int): Number of vertical axis (or floors) for the grid of the frame.
    @param storey_width (float): Width of the bays.
    @param storey_height (float): Height of the storeys.
    @param half_pz_height (np.ndarray, optional): Array of 1 dimension with half the height of the panel zone for each floor.
        The first floor should be 0 (no panel zone in the support). Defaults to np.array([]), e.g. no panel zone.
    @param origin (list, optional): List of two entry with the origin position. Defaults to [0, 0].
    @param first_hor_axis (int, optional): Number of the first pier. Defaults to 1.
    @param first_vert_axis (int, optional): Number of the first floor. Defaults to 1.
    @param show_plot (bool, optional): Option to show the plot of the nodes declared and initialised. Defaults to True.

    @exception NegativeValue: n_hor_axis needs to be a positive integer.
    @exception NegativeValue: n_vert_axis needs to be a positive integer.
    @exception NegativeValue: storey_width needs to be positive.
    @exception NegativeValue: storey_height needs to be positive.
    @exception WrongDimension: origin has a dimension of 2.
    @exception NegativeValue: first_hor_axis needs to be a positive integer.
    @exception NegativeValue: first_vert_axis needs to be a positive integer.
    @exception WrongDimension: size of half_pz_height needs to be equal to n_vert_axis, if different from 0.

    @returns list: List with the nodes declared.
    """
    if n_hor_axis < 1: raise NegativeValue()
    if n_vert_axis < 1: raise NegativeValue()
    if storey_width < 0: raise NegativeValue()
    if storey_height < 0: raise NegativeValue()
    if len(origin) != 2: raise WrongDimension()
    if first_hor_axis < 1: raise NegativeValue()
    if first_vert_axis < 1: raise NegativeValue()
    if np.size(half_pz_height) != 0 and np.size(half_pz_height) != n_vert_axis: raise WrongDimension()

    if np.size(half_pz_height) == 0: half_pz_height = np.zeros(n_vert_axis)
    node_array = []
    max_n_x = n_hor_axis + first_hor_axis - 1
    max_n_y = n_vert_axis + first_vert_axis - 1
    for xx in range(n_hor_axis):
        x_axis = xx + first_hor_axis
        for yy in range(n_vert_axis):
            y_axis = yy + first_vert_axis
            node_ID = GridIDConvention(x_axis, y_axis, max_n_x, max_n_y)
            node(node_ID, origin[0]+xx*storey_width, origin[1]+yy*storey_height+half_pz_height[yy])
            if y_axis == 1 and half_pz_height[yy] != 0: print("Warning: the nodes at the base have a panel zone height")
            node_array.append(node_ID)

    if show_plot:
        plot_nodes(node_array, "Frame geometry template with only nodes", True)
        plt.grid()

    return node_array


def DefineFrameNodesAndElementsSteelIShape(n_hor_axis: int, n_vert_axis: int, storey_width, storey_height,
    list_col: list, list_beam: list, geo_trans_ID: int, N_G = np.array([]), t_dp = np.array([]), L_b_col = np.array([]), L_b_beam = np.array([]),
    fix_support = True, show_plot = True, panel_zone = True):
    """
    WIP (Work In Progress). Function that declares and initialises the grid nodes of a frame and the members using steel I shape SpringBasedElements.
    WARNING: Current limit of the geometry: n_hor_axis and n_vert_axis < 10; if exceeded, there are problems with the IDs (ID limit is exceeded, ~2.2e9).
    WARNING: if the section of the columns change, the function does not account for the splacing. Each colum section is defined from floor to floor;
      if there is a change in the column section, it happens right after the panel zone (not realistic but good enough for predesign).
    WIP: Solve ID limit for large building need implementations (for example the use of a different ID convention or the use of the class IDGenerator).

    @param n_hor_axis (int): Number of horizontal axis (or piers) for the grid of the frame.
    @param n_vert_axis (int): Number of vertical axis (or floors) for the grid of the frame.
    @param storey_width (float): Width of the bays.
    @param storey_height (float): Height of the storeys.
    @param list_col (list(SteelIShape)): List with the sections of the columns for every floor.
    @param list_beam (list(SteelIShape)): List with the sections of the beams for every bay.
    @param geo_trans_ID (int): The geometric transformation (for more information, see OpenSeesPy documentation).
    @param N_G (np.ndarray, optional): Array of dimension 1 with the axial load for each column (starting at floor 2). Defaults to np.array([]), e.g. 0.
    @param t_dp (np.ndarray, optional): Array of dimension 1 with the doubler plate thickness for each bay's beam. Defaults to np.array([]), e.g. 0.
    @param L_b_col (np.ndarray, optional): Array of dimension 1 with the maxiaml unbraced lateral buckling length for each column. Defaults to np.array([]), e.g. -1.
```

```
    @param L_b_beam (np.ndarray, optional): Array of dimension 1 with the maxiaml unbraced lateral buckling length for each beam. Defaults to np.array([]), e.g. -1.
    @param fix_support (bool, optional): Option to fix the support of the frame. Defaults to True.
    @param show_plot (bool, optional): Option to show the plot of the nodes declared and initialised. Defaults to True.
    @param panel_zone (bool, optional): Option to add the panel zones in the model. Defaults to True.

    @exception WrongDimension: N_G dimension needs to be equal to n_vert_axis-1, if different from 0.
    @exception WrongDimension: t_dp dimension needs to be equal to n_vert_axis-1, if different from 0.
    @exception WrongDimension: L_b_col dimension needs to be equal to n_vert_axis-1, if different from 0.
    @exception WrongDimension: L_b_beam dimension needs to be equal to n_hor_axis-1, if different from 0.
    @exception WrongDimension: list_col dimension needs to be equal to n_vert_axis-1.
    @exception WrongDimension: list_beam dimension needs to be equal to n_vert_axis-1.
    @exception NegativeValue: geo_trans_ID needs to be a positive integer.

    @returns List: List with the element objects in the frame.
    """
    panel_zone = True
    if np.size(N_G) == 0: N_G = np.zeros(n_vert_axis-1)
    if np.size(t_dp) == 0: t_dp = np.zeros(n_vert_axis-1)
    if np.size(L_b_col) == 0: L_b_col = np.ones(n_vert_axis-1) * (-1.0)
    if np.size(L_b_beam) == 0: L_b_beam = np.ones(n_hor_axis-1) * (-1.0)

    if np.size(list_col) != n_vert_axis-1: raise WrongDimension()
    if np.size(list_beam) != n_vert_axis-1: raise WrongDimension()
    if np.size(N_G) != n_vert_axis-1: raise WrongDimension()
    if np.size(t_dp) != n_vert_axis-1: raise WrongDimension()
    if np.size(L_b_col) != n_vert_axis-1: raise WrongDimension()
    if np.size(L_b_beam) != n_hor_axis-1: raise WrongDimension()
    if geo_trans_ID < 1: raise NegativeValue()

    half_pz_height = np.zeros(n_vert_axis)
    if panel_zone:
        for ii, beam in enumerate(list_beam):
            half_pz_height[ii+1] = beam.d/2

    node_array = DefineFrameNodes(n_hor_axis, n_vert_axis, storey_width, storey_height, half_pz_height, [0, 0], 1, 1, False)

    beam_column_pzspring = [[], [], []]
    for xx in range(n_hor_axis):
        for yy in range(n_vert_axis):
            node_ID = node_array[xx*n_vert_axis + yy]
            if yy != 0:
                # Panel Zone
                if half_pz_height[yy] == 0:
                    col_j_node_ID = node_ID
                    beam_j_node_ID = node_ID
                else:
                    tmp_pz = PanelZoneSteelIShapeSkiadopoulos2021(node_ID, list_col[yy-1], list_beam[yy-1], geo_trans_ID, t_dp[yy-1])
                    tmp_pz.CreateMember()
                    col_j_node_ID = IDConvention(node_ID, 5, 1)
                    beam_j_node_ID = IDConvention(node_ID, 8, 1)
                    beam_column_pzspring[2].append(deepcopy(tmp_pz))

                # Column
                col_i_node_ID = node_array[xx*n_vert_axis + yy - 1]
                col_mat_i = OffsetNodeIDConvention(col_i_node_ID, "vertical", "i")
                col_mat_j = OffsetNodeIDConvention(col_j_node_ID, "vertical", "j")
                ele_ID = col_i_node_ID if panel_zone else -1
                tmp_col = SpringBasedElementModifiedIMKSteelIShape(col_i_node_ID, col_j_node_ID, list_col[yy-1], geo_trans_ID,
                    col_mat_i, col_mat_j, N_G[yy-1], L_b=L_b_col[yy-1], ele_ID = ele_ID)
                tmp_col.CreateMember()
                beam_column_pzspring[1].append(deepcopy(tmp_col))

                if xx != 0:
                    # Beam
                    if half_pz_height[yy] == 0:
                        beam_i_node_ID = node_array[(xx-1)*n_vert_axis + yy]
                    else:
                        beam_i_node_ID = IDConvention(node_array[(xx-1)*n_vert_axis + yy], 2, 1)
                    beam_mat_i = OffsetNodeIDConvention(beam_i_node_ID, "horizontal", "i")
                    beam_mat_j = OffsetNodeIDConvention(beam_j_node_ID, "horizontal", "j")
                    ele_ID = beam_i_node_ID if panel_zone else -1
                    tmp_beam = SpringBasedElementModifiedIMKSteelIShape(beam_i_node_ID, beam_j_node_ID, list_beam[yy-1], geo_trans_ID,
                        beam_mat_i, beam_mat_j, L_b=L_b_beam[xx-1], ele_ID = ele_ID)
                    tmp_beam.CreateMember()
                    beam_column_pzspring[0].append(deepcopy(tmp_beam))
            else:
                if fix_support: RigidSupport(node_ID)


```

```
    if show_plot:
        opsplt.plot_model("nodes", "elements")

    return beam_column_pzspring


def DefineSubassemblageNodes(beam_left_L_cl, beam_right_L_cl, col_top_L_cl, col_bottom_L_cl, depth_col, depth_beam,
    boundary_condition = True, show_plot = True):
    """
    Function that declares and initialises the grid nodes of an interior subassemblage. The panel zone geometry is defined by the two arguments
        depth_col and depth_beam.

    @param beam_left_L_cl (float): Centerline length of the left beam (excluding the panel zone).
    @param beam_right_L_cl (float): Centerline length of the right beam (excluding the panle zone).
    @param col_top_L_cl (float): Centerline length of the top column (excluding the panel zone).
    @param col_bottom_L_cl (float): Centerline length of the bottom column (excluding the panel zone).
    @param depth_col (float): Depth of the columns for the panel zone.
    @param depth_beam (float): Depth of the beams for the panel zone.
    @param boundary_condition (bool, optional): Option to set already the boundary condition (bottom column pinned, beams fix only y movement).
        Defaults to True.
    @param show_plot (bool, optional): Option to show the plot of the nodes declared and initialised. Defaults to True.

    @exception NegativeValue: beam_left_L_cl needs to be positive.
    @exception NegativeValue: beam_right_L_cl needs to be positive.
    @exception NegativeValue: col_top_L_cl needs to be positive.
    @exception NegativeValue: col_bottom_L_cl needs to be positive.
    @exception NegativeValue: depth_col needs to be positive.
    @exception NegativeValue: depth_beam needs to be positive.

    @returns list: List with the nodes declared.
    """
    # origin is the bottom left corner
    if beam_left_L_cl < 0: raise NegativeValue()
    if beam_right_L_cl < 0: raise NegativeValue()
    if col_top_L_cl < 0: raise NegativeValue()
    if col_bottom_L_cl < 0: raise NegativeValue()
    if depth_col < 0: raise NegativeValue()
    if depth_beam < 0: raise NegativeValue()

    node(12, 0.0, col_bottom_L_cl+depth_beam/2)
    node(21, beam_left_L_cl+depth_col/2, 0.0)
    node(22, beam_left_L_cl+depth_col/2, col_bottom_L_cl+depth_beam)
    node(23, beam_left_L_cl+depth_col/2, col_bottom_L_cl+depth_beam+col_top_L_cl)
    node(32, beam_left_L_cl+depth_col+beam_right_L_cl, col_bottom_L_cl+depth_beam/2)
    node_array = [12, 21, 22, 23, 32]

    if boundary_condition:
        fix(12, 0, 1, 0)
        fix(32, 0, 1, 0)
        fix(21, 1, 1, 0)

    if show_plot:
        plot_nodes(node_array, "Subassemblage geometry template with only nodes", True)
        plt.grid()

    return node_array


# def DefineRCSSubassemblage():
#     # WIP and experimental


```

8.17 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py File Reference

Classes

- class ConfMander1988Circ

  Class that stores funcions and material properties of a RC circular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.
- class ConfMander1988CircRCCircShape

Class that is the children of ConfMander1988Circ and combine the class RCCircShape (section) to retrieve the information needed.

- class ConfMander1988Rect

  Class that stores funcions and material properties of a RC rectangular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

- class ConfMander1988RectRCRectShape

  Class that is the children of ConfMander1988Rect and combine the class RCRectShape (section) to retrieve the information needed.

- class GMP1970

  Class that stores funcions and material properties of the vertical steel reinforcement bars with Giuffré, Menegotto and Pinto 1970 as the material model and the OpenSeesPy command type used to model it is Steel02.

- class GMP1970RCRectShape

  Class that is the children of GMP1970 and combine the class RCRectShape (section) to retrieve the information needed.

- class Gupta1999

  Class that stores funcions and material properties of a steel double symmetric I-shape profile with Gupta 1999 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.

- class Gupta1999SteelIShape

  Class that is the children of Gupta1999 and combine the class SteelIShape (section) to retrieve the information needed.

- class MaterialModels

  Parent abstract class for the storage and manipulation of a material model's information (mechanical and geometrical parameters, etc) and initialisation in the model.

- class ModifiedIMK

  Class that stores funcions and material properties of a steel double symmetric I-shape profile with modified Ibarra-↩ Medina-Krawinkler as the material model for the nonlinear springs and the OpenSeesPy command type used to model it is Bilin.

- class ModifiedIMKSteelIShape

  Class that is the children of ModifiedIMK and combine the class SteelIShape (section) to retrieve the information needed.

- class Skiadopoulos2021

  Class that stores funcions and material properties of a steel double symmetric I-shape profile with Skiadopoulos 2021 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.

- class Skiadopoulos2021RCS

  WIP: Class that is the children of Skiadopoulos2021 and it's used for the panel zone spring in a RCS (RC column continous, Steel beam).

- class Skiadopoulos2021SteelIShape

  Class that is the children of Skiadopoulos2021 and combine the class SteelIShape (section) to retrieve the information needed.

- class UnconfMander1988

  Class that stores funcions and material properties of a RC rectangular or circular section with Mander 1988 as the material model for the unconfined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

- class UnconfMander1988RCCircShape

  Class that is the children of UnconfMander1988 and combine the class RCCircShape (section) to retrieve the information needed.

- class UnconfMander1988RCRectShape

  Class that is the children of UnconfMander1988 and combine the class RCRectShape (section) to retrieve the information needed.

- class UniaxialBilinear

  Class that stores funcions and material properties of a simple uniaxial bilinear model with the OpenSeesPy command type used to model it is Steel01.

- class UniaxialBilinearSteelIShape

Class that is the children of UniaxialBilinear and combine the class SteelIShape (section) to retrieve the information needed.
- class UVC

    Class that stores funcions and material properties of a steel profile or reinforcing bar with Updated Voce-Chaboche as the material model and the OpenSeesPy command type used to model it is UVCuniaxial.
- class UVCCalibrated

    Class that is the children of UVC that retrieve calibrated parameters from UVC_calibrated_parameters.txt.
- class UVCCalibratedRCCircShape

    Class that is the children of UVCCalibrated and combine the class RCCircShape (section) to retrieve the information needed.
- class UVCCalibratedRCRectShape

    Class that is the children of UVCCalibrated and combines the class RCRectShape (section) to retrieve the information needed.
- class UVCCalibratedSteelIShapeFlange

    Class that is the children of UVCCalibrated and combine the class SteelIShape (section) to retrieve the information needed for the material model of the flange (often used fo the entire section).
- class UVCCalibratedSteelIShapeWeb

    Class that is the children of UVCCalibrated and combine the class SteelIShape (section) to retrieve the information needed for the material model of the web.

Namespaces

- namespace MaterialModels

    Module for the material models.

Functions

- def Concrete01Funct (fc, ec, fpcu, ecu, discretized_eps)

    Function with the equation of the curve of the Concrete01 model.
- def Concrete04Funct (fc, discretized_eps, ec, Ec)

    Function with the equation of the curve of the confined and unconfined concrete (Popovics 1973).
- def PlotConcrete01 (fc, ec, fpcu, ecu, ax, ID=0)

    Function that plots the Concrete01 stress-strain curve.
- def PlotConcrete04 (fc, Ec, ec, ecu, str Type, ax, ID=0)

    Function that plots the confined/unconfined Concrete04 stress-strain curve.

8.18 MaterialModels.py

Go to the documentation of this file.

```
"""
Module for the material models.
Carmine Schipani, 2021
"""

from openseespy.opensees import *
import matplotlib.pyplot as plt
import numpy as np
import os
import math
from abc import abstractmethod
from copy import copy, deepcopy
from OpenSeesPyAssistant.Section import *
from OpenSeesPyAssistant.DataManagement import *
from OpenSeesPyAssistant.ErrorHandling import *
from OpenSeesPyAssistant.Units import *

```

```

class MaterialModels(DataManagement):
    """
    Parent abstract class for the storage and manipulation of a material model's information (mechanical
    and geometrical parameters, etc) and initialisation in the model.

    @param DataManagement: Parent abstract class.
    """
    @abstractmethod
    def CheckApplicability(self):
        """
        Abstract function used to check the applicability of the material model.
        """
        pass


class ModifiedIMK(MaterialModels):
    """
    Class that stores funcions and material properties of a steel double symmetric I-shape profile
        with modified Ibarra-Medina-Krawinkler as the material model for the nonlinear springs and the OpenSeesPy command type used to model it is Bilin.
    The default values are valid for a simple cantelever.
    For more information about the empirical model for the computation of the parameters, see Lignos Krawinkler 2011.
    The parameter 'n' is used as global throughout the SteelIShape sections to optimise the program (given the fact that is constant everytime).

    @param MaterialModels: Parent abstract class.
    """
    global n
    n = 10.0

    def __init__(self, ID: int, Type: str, d, bf, tf, tw, h_1, Iy_mod, iz, E, Fy, Npl, My, L,
        N_G = 0, K_factor = 3, L_0 = -1, L_b = -1, Mc = -1, K = -1, theta_u = -1, safety_factors = False):
        """
        Constructor of the class. Every argument that is optional and is initialised as -1, will be computed in this class.

        @param ID (int): ID of the material model.
        @param Type (str): Type of the section. It can be 'Col' for column or 'Beam' for beams.
        @param d (float): Depth of the section.
        @param bf (float): Flange's width of the section
        @param tf (float): Flange's thickness of the section
        @param tw (float): Web's thickness of the section
        @param h_1 (float): Depth excluding the flange's thicknesses and the weld fillets.
        @param Iy_mod (float): n modified moment of inertia (strong axis)
        @param iz (float): Radius of gyration (weak axis).
        @param E (float): Young modulus.
        @param Fy (float): Yield strength.
        @param Npl (float): Maximal vertical axial load.
        @param My (float): Yielding moment.
        @param L (float): Effective length of the element associated with this section.
            If the panel zone is present, exclude its dimension.
        @param N_G (float, optional): Gravity axial load. Defaults to 0.
        @param K_factor (float, optional): Rigidity factor. Defaults to 3 (assuming cantilever).
        @param L_0 (float, optional): Position of the inflection point.
            Defaults to -1, e.g. computed as the total length, assuming cantilever.
        @param L_b (float, optional): Maximal unbraced lateral torsional buckling length.
            Defaults to -1, e.g. computed as the total length, assuming cantilever with no bracing support.
        @param Mc (float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
        @param K (float, optional): Residual strength ratio. Defaults to -1, e.g. computed in ComputeK.
        @param theta_u (float, optional): Ultimate rotation. Defaults to -1, e.g. computed in ComputeTheta_u.
        @param safety_factors (bool, optional): Safety factors used if standard mechanical parameters are used (not test results). Defaults to False.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception WrongArgument: Type needs to be 'Col' or 'Beam'.
        @exception NegativeValue: d needs to be positive.
        @exception NegativeValue: bf needs to be positive.
        @exception NegativeValue: tf needs to be positive.
        @exception NegativeValue: tw needs to be positive.
        @exception NegativeValue: h_1 needs to be positive.
        @exception NegativeValue: Iy_mod needs to be positive.
        @exception NegativeValue: iz needs to be positive.
        @exception NegativeValue: E needs to be positive.
        @exception NegativeValue: Fy needs to be positive.
        @exception NegativeValue: Npl needs to be positive.
        @exception NegativeValue: My needs to be positive.
        @exception NegativeValue: L needs to be positive.
        @exception NegativeValue: N_G needs to be positive.
        @exception NegativeValue: L_0 needs to be positive if different from -1.
        @exception NegativeValue: L_b needs to be positive if different from -1.
```

```
        @exception NegativeValue: Mc needs to be positive if different from -1.
        @exception NegativeValue: K needs to be positive if different from -1.
        @exception NegativeValue: theta_u needs to be positive if different from -1.
        @exception InconsistentGeometry: h_1 can't be bigger than d
        @exception MemberFailure: N_G can't be bigger than Npl (section failure).
        @exception InconsistentGeometry: L_0 can't be bigger than L
        """
        # Check
        if ID < 0: raise NegativeValue()
        if Type != "Beam" and Type != "Col": raise WrongArgument()
        if d < 0: raise NegativeValue()
        if bf < 0: raise NegativeValue()
        if tf < 0: raise NegativeValue()
        if tw < 0: raise NegativeValue()
        if h_1 < 0: raise NegativeValue()
        if Iy_mod < 0: raise NegativeValue()
        if iz < 0: raise NegativeValue()
        if E < 0: raise NegativeValue()
        if Fy < 0: raise NegativeValue()
        if Npl < 0: raise NegativeValue()
        if My < 0: raise NegativeValue()
        if L < 0: raise NegativeValue()
        if N_G < 0: raise NegativeValue()
        if L_0 != -1 and L_0 < 0: raise NegativeValue()
        if L_b != -1 and L_b < 0: raise NegativeValue()
        if Mc != -1 and Mc < 0: raise NegativeValue()
        if K != -1 and K < 0: raise NegativeValue()
        if theta_u != -1 and theta_u < 0: raise NegativeValue()
        if h_1 > d: raise InconsistentGeometry()
        if N_G > Npl: raise MemberFailure()
        if L_0 > L: raise InconsistentGeometry()

        # Arguments
        self.TypeType = Type
        self.IDID = ID
        self.dd = d
        self.bfbf = bf
        self.tftf = tf
        self.twtw = tw
        self.h_1h_1 = h_1
        self.Iy_modIy_mod = Iy_mod
        self.iziz = iz
        self.EE = E
        self.FyFy = Fy
        self.NplNpl = Npl
        self.MyMy = My
        self.LL = L
        self.N_GN_G = N_G
        self.K_factorK_factor = K_factor
        self.L_0L_0 = L if L_0 == -1 else L_0
        self.L_bL_b = L if L_b == -1 else L_b

        # Initialized the parameters that are dependent from others
        self.section_name_tagsection_name_tag = "None"
        self.InitializedInitialized = False
        if safety_factors:
            self.gamma_rmgamma_rm = 1.25
            self.prob_factorprob_factor = 1.15
        else:
            self.gamma_rmgamma_rm = 1.0
            self.prob_factorprob_factor = 1.0
        self.ReInitReInit(Mc, K, theta_u)


    # Methods
    def ReInit(self, Mc = -1, K = -1, theta_u = -1):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param Mc (float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
        @param K (float, optional): Residual strength ratio. Defaults to -1, e.g. computed in ComputeK.
        @param theta_u (float, optional): Ultimate rotation. Defaults to -1, e.g. computed in ComputeTheta_u.
        """
        # Precompute some members
        self.My_starMy_star = self.ComputeMyStarComputeMyStar()

        # Arguments
        self.McMc = self.ComputeMcComputeMc() if Mc == -1 else Mc
        self.KK = self.ComputeKComputeK() if K == -1 else K
        self.theta_utheta_u = self.ComputeTheta_uComputeTheta_u() if theta_u == -1 else theta_u

        # Check applicability
        self.CheckApplicabilityCheckApplicabilityCheckApplicability()

```

```
        # Members
        self.KeKe = self.ComputeKeComputeKe()
        self.theta_ytheta_y = self.ComputeTheta_yComputeTheta_y()
        self.theta_ptheta_p = self.ComputeTheta_pComputeTheta_p()
        self.theta_pctheta_pc = self.ComputeTheta_pcComputeTheta_pc()
        self.McMyMcMy = self.McMc/self.My_starMy_star
        self.rate_detrate_det = self.ComputeRefEnergyDissipationCapComputeRefEnergyDissipationCap()
        self.aa = self.ComputeaComputea()
        self.a_sa_s = self.Computea_sComputea_s()
        if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag = self.section_name_tagsection_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "ModifiedIMK"], # Tag for differentiating different data
            ["ID", self.IDID],
            ["section_name_tag", self.section_name_tagsection_name_tag],
            ["Type", self.TypeType],
            ["d", self.dd],
            ["bf", self.bfbf],
            ["tf", self.tftf],
            ["tw", self.twtw],
            ["h_1", self.h_1h_1],
            ["Iy_mod", self.Iy_modIy_mod],
            ["iz", self.iziz],
            ["E", self.EE],
            ["Fy", self.FyFy],
            ["L", self.LL],
            ["N_G", self.N_GN_G],
            ["K_factor", self.K_factorK_factor],
            ["Ke", self.KeKe],
            ["L_0", self.L_0L_0],
            ["L_b", self.L_bL_b],
            ["gamma_rm", self.gamma_rmgamma_rm],
            ["prob_factor", self.prob_factorprob_factor],
            ["Npl", self.NplNpl],
            ["My", self.MyMy],
            ["My_star", self.My_starMy_star],
            ["Mc", self.McMc],
            ["McMy", self.McMyMcMy],
            ["K", self.KK],
            ["theta_y", self.theta_ytheta_y],
            ["theta_p", self.theta_ptheta_p],
            ["theta_pc", self.theta_pctheta_pc],
            ["theta_u", self.theta_utheta_u],
            ["rate_det", self.rate_detrate_det],
            ["a", self.aa],
            ["a_s", self.a_sa_s],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self, plot = False, block = False):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command ’plt.show()’ (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        """
        Mr = self.KK*self.My_starMy_star
        theta_p_plot = self.theta_ptheta_p
        if self.theta_ptheta_p > self.theta_utheta_u-self.theta_ytheta_y:
            theta_p_plot = self.theta_utheta_u-self.theta_ytheta_y
        theta_r = self.theta_ytheta_y + theta_p_plot + self.theta_pctheta_pc*(1.0-Mr/self.McMc)
        if theta_r > self.theta_utheta_u:
            theta_r = self.theta_utheta_u
            Mr = self.McMc*(1.0-1.0/self.theta_pctheta_pc*(self.theta_utheta_u-self.theta_ytheta_y-theta_p_plot))

        print("")
        print("Requested info for Modified IMK (Ibarra-Medina-Krawinkler) material model Parameters, ID = {}".format(self.IDID))
        print("Section associated: {}".format(self.section_name_tagsection_name_tag))
        print(’theta y = {}’.format(self.theta_ytheta_y))
        print(’theta p = {}’.format(self.theta_ptheta_p))
        print(’theta r = {}’.format(theta_r))
        print(’theta pc = {}’.format(self.theta_pctheta_pc))
        print(’theta u = {}’.format(self.theta_utheta_u))
```

```
        print('My star = {} kNm'.format(self.My_starMy_star/kNm_unit))
        print('Mc = {} kNm'.format(self.McMc/kNm_unit))
        print('Mr = {} kNm'.format(Mr/kNm_unit))
        print('a = {} '.format(self.aa))
        print('as = {} '.format(self.a_sa_s))
        print('lambda (deterioration rate) = {} '.format(self.rate_detrate_det))
        print("")

        if plot:
            # Data for plotting
            x_axis = np.array([0.0, self.theta_ytheta_y, self.theta_ytheta_y + theta_p_plot, theta_r,
    self.theta_utheta_u, self.theta_utheta_u])
            x_axis2 = np.array([self.theta_ytheta_y + theta_p_plot, self.theta_ytheta_y + theta_p_plot
    + self.theta_pctheta_pc])
            y_axis = np.array([0.0, self.My_starMy_star, self.McMc, Mr, Mr, 0.0])/kNm_unit
            y_axis2 = np.array([self.McMc, 0.0])/kNm_unit

            fig, ax = plt.subplots()
            ax.plot(x_axis, y_axis, 'k-')
            ax.plot(x_axis2, y_axis2, 'k--')

            ax.set(xlabel='Rotation [rad]', ylabel='Moment [kNm]',
                title='Modified IMK deterioration model (ID={})'.format(self.IDID))
            ax.grid()

            if block:
                plt.show()


    def CheckApplicability(self):
        """
        Implementation of the homonym abstract method.
        See parent class MaterialModels for detailed information.
        """
        Check = True
        if self.TypeType == "Beam":
            if self.dd/self.twtw < 20 or self.dd/self.twtw > 55:
                Check = False
                print("The d/tw check was not fullfilled")
            if self.L_bL_b/self.iziz < 20 or self.L_bL_b/self.iziz > 80:
                Check = False
                print("The Lb/iz check was not fullfilled")
            if self.bfbf/2/self.tftf < 4 or self.bfbf/2/self.tftf  > 8:
                Check = False
                print("The bf/2/tf check was not fullfilled")
            if self.LL/self.dd < 2.5 or self.LL/self.dd > 7:
                Check = False
                print("The check L/d was not fullfilled")
            if self.dd < 102*mm_unit or self.dd > 914*mm_unit:
                Check = False
                print("The d check was not fullfilled")
            if self.FyFy < 240*MPa_unit or self.FyFy > 450*MPa_unit:
                Check = False
                print("The Fy check was not fullfilled")
        else:
            if self.h_1h_1/self.twtw < 3.71 or self.dd/self.twtw > 57.5:
                Check = False
                print("The h1/tw check was not fullfilled")
            if self.L_bL_b/self.iziz < 38.4 or self.L_bL_b/self.iziz > 120:
                Check = False
                print("The Lb/iz check was not fullfilled")
            if self.N_GN_G/self.NplNpl < 0 or self.N_GN_G/self.NplNpl > 0.75:
                Check = False
                print("The NG/Npl check was not fullfilled")
        if not Check:
            print("The validity of the equations is not fullfilled.")
            print("!!!!!!! WARNING !!!!!!! Check material model of Modified IMK, ID=", self.IDID)
            print("")

    def ComputeKe(self):
        """
        Method that computes the elastic stiffness.

        @returns float: The stiffness
        """
        return self.K_factorK_factor*n*self.EE*self.Iy_modIy_mod/self.LL

    def Computea(self):
        """
        Method that computes the strain hardening ratio with the n modification.

        @returns float: Strain hardening ratio.
        """
        # strain hardening ratio of spring
        return (n+1.0)*self.My_starMy_star*(self.McMyMcMy-1.0)/(self.KeKe*self.theta_ptheta_p)

    def Computea_s(self):
```

```
        """
        Method that computes the modified strain hardening ratio for the spring.
        For more info see Ibarra & Krawinkler 2005.

        @returns float: Strain hardening ratio.
        """
        return self.aa/(1.0+n*(1.0-self.aa))

    def ComputeMyStar(self):
        """
        Method that computes the effective yield moment.
        For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

        @returns float: Effective yield moment.
        """
        if self.TypeType == "Beam":
            return self.prob_factorprob_factor*self.MyMy*self.gamma_rmgamma_rm*1.1
        else:
            if self.N_GN_G/self.NplNpl > 0.2:
                return
1.15*self.prob_factorprob_factor*self.MyMy*self.gamma_rmgamma_rm*(1-self.N_GN_G/self.NplNpl)*9.0/8.0
            else:
                return
1.15*self.prob_factorprob_factor*self.MyMy*self.gamma_rmgamma_rm*(1-self.N_GN_G/2.0/self.NplNpl)

    def ComputeMc(self):
        """
        Method that computes the capping moment.
        For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

        @returns float: Capping moment.
        """
        if self.TypeType == "Beam":
            return self.My_starMy_star*1.11
            # For RBS: My_star*1.09
        else:
            tmp =
12.5*(self.h_1h_1/self.twtw)**(-0.2)*(self.L_bL_b/self.iziz)**(-0.4)*(1-self.N_GN_G/self.NplNpl)**0.4
            return max(min(1.3, tmp), 1.0)*self.My_starMy_star

    def ComputeK(self):
        """
        Method that computes the residual strength ratio.
        For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

        @returns float: Residual strength ratio.
        """
        if self.TypeType == "Beam":
            return 0.4
        else:
            tmp = 0.5-0.4*self.N_GN_G/self.NplNpl
            return max(tmp, 0)

    def ComputeTheta_y(self):
        """
        Method that computes the yield rotation.
        For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

        @returns float: Yield rotation.
        """
        return self.My_starMy_star/self.KeKe*(n+1)

    def ComputeTheta_p(self):
        """
        Method that computes the plastic rotation.
        For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

        @returns float: Plastic rotation.
        """
        if self.TypeType == "Beam":
            if self.dd < 533.0*mm_unit:
                return
0.0865*(self.h_1h_1/self.twtw)**(-0.365)*(self.bfbf/2.0/self.tftf)**(-0.14)*(self.L_0L_0/self.dd)**(0.34)*(self.dd/(53
            else:
                return
0.318*(self.h_1h_1/self.twtw)**(-0.550)*(self.bfbf/2.0/self.tftf)**(-0.345)*(self.L_0L_0/self.dd)**(0.090)*(self.L_bL_
                # With RBS: ...
        else:
            tmp =
294.0*(self.h_1h_1/self.twtw)**(-1.7)*(self.L_bL_b/self.iziz)**(-0.7)*(1.0-self.N_GN_G/self.NplNpl)**(1.6)
# *(self.E/self.Fy/gamma_rm)**(0.2) # EC8
            if tmp > 0.2:
                tmp = 0.2
            # if tmp > self.theta_u-self.theta_y:
            #     tmp = (self.theta_u-self.theta_y)*0.799 # convergence issue
            return tmp

```

```
    def ComputeTheta_pc(self):
        """
        Method that computes the post capping rotation.
        For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

        @returns float: Post capping rotation.
        """
        if self.TypeType == "Beam":
            if self.dd < 533.0*mm_unit:
                return
5.63*(self.h_1h_1/self.twtw)**(-0.565)*(self.bfbf/2.0/self.tftf)**(-0.800)*(self.dd/(533.0*mm_unit))**(-0.280)*(self.Fy
            else:
                return
7.50*(self.h_1h_1/self.twtw)**(-0.610)*(self.bfbf/2.0/self.tftf)**(-0.710)*(self.L_bL_b/self.iziz)**(-0.110)*(self.dd/
                # With RBS: ...
        else:
            tmp =
90.0*(self.h_1h_1/self.twtw)**(-0.8)*(self.L_bL_b/self.iziz)**(-0.8)*(1.0-self.N_GN_G/self.NplNpl)**(2.5)
# *(self.E/self.Fy/gamma_rm)**(0.07) # EC8
            return min(tmp, 0.3)

    def ComputeTheta_u(self):
        """
        Method that computes the ultimate rotation.
        For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

        @returns float: Ultimate rotation.
        """
        if self.TypeType == "Beam":
            return 0.2
        else:
            return 0.15

    def ComputeRefEnergyDissipationCap(self):
        """
        Method that computes the reference energy dissipation capacity.
        For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

        @returns float: Reference energy dissipation capacity.
        """
        if self.TypeType == "Beam":
            if self.dd < 533.0*mm_unit:
                return
495.0*(self.h_1h_1/self.twtw)**(-1.34)*(self.bfbf/2.0/self.tftf)**(-0.595)*(self.FyFy/(355.0*MPa_unit))**(-0.360)
            else:
                return
536.0*(self.h_1h_1/self.twtw)**(-1.26)*(self.bfbf/2.0/self.tftf)**(-0.525)*(self.L_bL_b/self.iziz)**(-0.130)*(self.FyFy
                # With RBS: ...
        else:
            if self.N_GN_G/self.NplNpl > 0.35:
                tmp =
268000.0*(self.h_1h_1/self.twtw)**(-2.30)*(self.L_bL_b/self.iziz)**(-1.130)*(1.0-self.N_GN_G/self.NplNpl)**(1.19)
                return min(tmp, 3.0)
            else:
                tmp =
25000.0*(self.h_1h_1/self.twtw)**(-2.14)*(self.L_bL_b/self.iziz)**(-0.53)*(1.0-self.N_GN_G/self.NplNpl)**(4.92)
                return min(tmp, 3.0)


    def Bilin(self):
        """
        Generate the material model Bilin (Modified IMK) using the computed parameters.
        See _Bilin function for more information.
        """
        _Bilin(self.IDID, self.KeKe, self.a_sa_s, self.My_starMy_star, self.theta_ptheta_p,
self.theta_pctheta_pc, self.KK, self.theta_utheta_u, self.rate_detrate_det)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


class ModifiedIMKSteelIShape(ModifiedIMK):
    """
    Class that is the children of ModifiedIMK and combine the class SteelIShape (section) to retrieve
the information needed.

    @param ModifiedIMK: Parent class.
    """
    def __init__(self, ID, section: SteelIShape, N_G = 0, K_factor = 3, L_0 = -1, L_b = -1, Mc = -1, K
= -1, theta_u = -1, safety_factors = False):
        """
        Constructor of the class. It passes the arguments into the parent class to generate the
combination of the parent class
            and the section class SteelIShape.
        Every argument that is optional and is initialised as -1, will be computed in this class.
        The copy of the section passed is stored in the member variable self.sectionsection.

        @param ID (int): ID of the material model.
```

```
        @param section (SteelIShape): Object that store informations for a steel I shpae section.
        @param N_G (float, optional): Gravity axial load. Defaults to 0.
        @param K_factor (float, optional): Rigidity factor. Defaults to 3 (assuming cantilever).
        @param L_0 (float, optional): Position of the inflection point.
            Defaults to -1, e.g. computed as the total length, assuming cantilever.
        @param L_b (float, optional):Maximal unbraced lateral torsional buckling length.
            Defaults to -1, e.g computed as the total length, assuming cantilever with no bracing support.
        @param Mc (float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
        @param K (float, optional): Residual strength ratio. Defaults to -1, e.g. computed in ComputeK.
        @param theta_u (float, optional): Ultimate rotation. Defaults to -1, e.g. computed in ComputeTheta_u.
        @param safety_factors (bool, optional): Safety factors used if standard mechanical parameters are used (not test results). Defaults to False.
        """
        self.sectionsection = deepcopy(section)
        super().__init__(ID, section.Type, section.d, section.bf, section.tf, section.tw, section.h_1,
            section.Iy_mod, section.iz, section.E, section.Fy, section.Npl, section.My, section.L, N_G,
            K_factor, L_0, L_b, Mc, K, theta_u, safety_factors)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()


class Gupta1999(MaterialModels):
    """
    Class that stores funcions and material properties of a steel double symmetric I-shape profile
        with Gupta 1999 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.
    The material model is valid only if the column is continuous.
    For more information about the empirical model for the computation of the parameters, see Gupta 1999.

    @param MaterialModels: Parent abstract class.
    """
    def __init__(self, ID: int, d_c, bf_c, tf_c, I_c, d_b, tf_b, Fy, E, t_p,
        t_dp = 0.0, a_s = 0.03, pinchx = 0.25, pinchy = 0.75, dmg1 = 0.0, dmg2 = 0.0, beta = 0.0, safety_factor = False):
        """
        Constructor of the class.

        @param ID (int): Unique material model ID.
        @param d_c (float): Column depth.
        @param bf_c (float): Column flange width.
        @param tf_c (float): Column flange thickness.
        @param I_c (float): Column moment of inertia (strong axis).
        @param d_b (float): Beam depth.
        @param tf_b (float): Beam flange thickness.
        @param Fy (float): Yield strength (if assume continous column, Fy of the web).
        @param E (float): Young modulus.
        @param t_p (float): Panel zone thickness.
        @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
        @param a_s (float, optional): Strain hardening. Defaults to 0.03.
        @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25.
        @param pinchy (float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75.
        @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
        @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
        @param beta (float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
        @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: d_c needs to be positive.
        @exception NegativeValue: bf_c needs to be positive.
        @exception NegativeValue: tf_c needs to be positive.
        @exception NegativeValue: d_b needs to be positive.
        @exception NegativeValue: tf_b needs to be positive.
        @exception NegativeValue: Fy needs to be positive.
        @exception NegativeValue: E needs to be positive.
        @exception NegativeValue: t_p needs to be positive.
        @exception NegativeValue: a_s needs to be positive.
        """
        # Check
        if ID < 1: raise NegativeValue()
        if d_c < 0: raise NegativeValue()
        if bf_c < 0: raise NegativeValue()
        if tf_c < 0: raise NegativeValue()
        if d_b < 0: raise NegativeValue()
        if tf_b < 0: raise NegativeValue()
        if Fy < 0: raise NegativeValue()
        if E < 0: raise NegativeValue()
        if t_p < 0: raise NegativeValue()
        if a_s < 0: raise NegativeValue()
```

```
        # Arguments
        self.IDID = ID
        self.d_cd_c = d_c
        self.bf_cbf_c = bf_c
        self.tf_ctf_c = tf_c
        self.I_cI_c = I_c
        self.d_bd_b = d_b
        self.tf_btf_b = tf_b
        self.FyFy = Fy
        self.EE = E
        self.t_pt_p = t_p
        self.t_dpt_dp = t_dp
        self.a_sa_s = a_s
        self.pinchxpinchx = pinchx
        self.pinchypinchy = pinchy
        self.dmg1dmg1 = dmg1
        self.dmg2dmg2 = dmg2
        self.betabeta = beta
        if safety_factor:
            self.RyRy = 1.2
        else:
            self.RyRy = 1.0

        # Initialized the parameters that are dependent from others
        self.beam_section_name_tagbeam_section_name_tag = "None"
        self.col_section_name_tagcol_section_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit()


    # Methods
    def ReInit(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        # Check applicability
        self.CheckApplicabilityCheckApplicabilityCheckApplicability()

        # Members
        if self.beam_section_name_tagbeam_section_name_tag != "None":
self.beam_section_name_tagbeam_section_name_tag = self.beam_section_name_tagbeam_section_name_tag + " (modified)"
        if self.col_section_name_tagcol_section_name_tag != "None":
self.col_section_name_tagcol_section_name_tag = self.col_section_name_tagcol_section_name_tag + " (modified)"

        # Trilinear Parameters
        self.t_pzt_pz = self.t_pt_p + self.t_dpt_dp
        self.VyVy = 0.55 * self.FyFy * self.RyRy * self.d_cd_c * self.t_pzt_pz # Yield Shear
        self.GG = self.EE/(2.0 * (1.0 + 0.30)) # Shear Modulus
        self.KeKe = 0.95 * self.GG * self.t_pzt_pz * self.d_cd_c # Elastic Stiffness
        self.KpKp = 0.95 * self.GG * self.bf_cbf_c * (self.tf_ctf_c * self.tf_ctf_c) / self.d_bd_b # Plastic Stiffness

        # Define Trilinear Equivalent Rotational Spring
        # Yield point for Trilinear Spring at gamma1_y
        self.gamma1_ygamma1_y = self.VyVy / self.KeKe
        self.M1yM1y = self.gamma1_ygamma1_y * (self.KeKe * self.d_bd_b)
        # Second Point for Trilinear Spring at 4 * gamma1_y
        self.gamma2_ygamma2_y = 4.0 * self.gamma1_ygamma1_y
        self.M2yM2y = self.M1yM1y + (self.KpKp * self.d_bd_b) * (self.gamma2_ygamma2_y - self.gamma1_ygamma1_y)
        # Third Point for Trilinear Spring at 100 * gamma1_y
        self.gamma3_ygamma3_y = 100.0 * self.gamma1_ygamma1_y
        self.M3yM3y = self.M2yM2y + (self.a_sa_s * self.KeKe * self.d_bd_b) * (self.gamma3_ygamma3_y - self.gamma2_ygamma2_y)

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "Gupta1999"], # Tag for differentiating different data
            ["ID", self.IDID],
            ["beam_section_name_tag", self.beam_section_name_tagbeam_section_name_tag],
            ["col_section_name_tag", self.col_section_name_tagcol_section_name_tag],
            ["d_c", self.d_cd_c],
            ["bf_c", self.bf_cbf_c],
            ["tf_c", self.tf_ctf_c],
            ["I_c", self.I_cI_c],
            ["d_b", self.d_bd_b],
```

```
            ["tf_b", self.tf_btf_b],
            ["Fy", self.FyFy],
            ["E", self.EE],
            ["G", self.GG],
            ["t_p", self.t_pt_p],
            ["t_dp", self.t_dpt_dp],
            ["t_pz", self.t_pzt_pz],
            ["a_s", self.a_sa_s],
            ["pinchx", self.pinchxpinchx],
            ["pinchy", self.pinchypinchy],
            ["dmg1", self.dmg1dmg1],
            ["dmg2", self.dmg2dmg2],
            ["beta", self.betabeta],
            ["Ry", self.RyRy],
            ["Vy", self.VyVy],
            ["Ke", self.KeKe],
            ["Kp", self.KpKp],
            ["gamma1_y", self.gamma1_ygamma1_y],
            ["M1y", self.M1yM1y],
            ["gamma2_y", self.gamma2_ygamma2_y],
            ["M2y", self.M2yM2y],
            ["gamma3_y", self.gamma3_ygamma3_y],
            ["M3y", self.M3yM3y],
            ["Initialized", self.InitializedInitialized]]

    def ShowInfo(self, plot = False, block = False):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to
    False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
    of the program everytime that a plot should pop up). Defaults to False.
        """
        print("")
        print("Requested info for Gupta 1999 material model Parameters, ID = {}".format(self.IDID))
        print("Sections associated, column: {}
    ".format(self.col_section_name_tagcol_section_name_tag))
        print("Sections associated, beam: {}
    ".format(self.beam_section_name_tagbeam_section_name_tag))
        print("gamma1_y = {} rad".format(self.gamma1_ygamma1_y))
        print("gamma2_y = {} rad".format(self.gamma2_ygamma2_y))
        print("gamma3_y = {} rad".format(self.gamma3_ygamma3_y))
        print("M1y = {} kNm".format(self.M1yM1y/kNm_unit))
        print("M2y = {} kNm".format(self.M2yM2y/kNm_unit))
        print("M3y = {} kNm".format(self.M3yM3y/kNm_unit))
        print("")

        if plot:
            # Data for plotting
            # Last point for plot
            gamma3_y_plot = 10.0 * self.gamma1_ygamma1_y
            M3y_plot = self.M2yM2y + (self.a_sa_s * self.KeKe * self.d_bd_b) * (gamma3_y_plot -
    self.gamma2_ygamma2_y)

            x_axis = np.array([0.0, self.gamma1_ygamma1_y, self.gamma2_ygamma2_y, gamma3_y_plot])
            y_axis = np.array([0.0, self.M1yM1y, self.M2yM2y, M3y_plot])/kNm_unit

            fig, ax = plt.subplots()
            ax.plot(x_axis, y_axis, 'k-')

            ax.set(xlabel='Rotation [rad]', ylabel='Moment [kNm]',
                title='Gupta 1999 material model (ID={})'.format(self.IDID))
            ax.grid()

            if block:
                plt.show()


    def CheckApplicability(self):
        """
        Implementation of the homonym abstract method.
        See parent class MaterialModels for detailed information.
        """
        Check = True
        # No checks
        if not Check:
            print("The validity of the equations is not fullfilled.")
            print("!!!!!!! WARNING !!!!!!! Check material model of Gupta 1999, ID=", self.IDID)
            print("")


    def Hysteretic(self):
        """
        Generate the material model Hysteretic (Gupta 1999) using the computed parameters.
        See _Hysteretic function for more information.
```

```
        """
        _Hysteretic(self.IDID, self.M1yM1y, self.gamma1_ygamma1_y, self.M2yM2y, self.gamma2_ygamma2_y, self.M3yM3y, self.gamma3_ygamma3_y,
            self.pinchxpinchx, self.pinchypinchy, self.dmg1dmg1, self.dmg2dmg2, self.betabeta)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


class Gupta1999SteelIShape(Gupta1999):
    """
    Class that is the children of Gupta1999 and combine the class SteelIShape (section) to retrieve the information needed.

    @param Gupta1999: Parent class.
    """
    def __init__(self, ID: int, col: SteelIShape, beam: SteelIShape,
        t_dp = 0.0, a_s = 0.03, pinchx = 0.25, pinchy = 0.75, dmg1 = 0.0, dmg2 = 0.0, beta = 0.0, safety_factor = False):
        """
        Constructor of the class. It passes the arguments into the parent class to generate the combination of the parent class
            and the section class SteelIShape.
        The copy of the sections (col and beam) passed is stored in the member variable self.section.

        @param ID (int): Unique material model ID.
        @param col (SteelIShape): SteelIShape column section object.
        @param beam (SteelIShape): SteelIShape beam section object.
        @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
        @param a_s (float, optional): Strain hardening. Defaults to 0.03.
        @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25.
        @param pinchy (float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75
        @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
        @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
        @param beta (float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
        @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.
        """
        self.colcol = deepcopy(col)
        self.beambeam = deepcopy(beam)
        super().__init__(ID, col.d, col.bf, col.tf, col.Iy, beam.d, beam.tf, col.Fy_web, col.E, col.tw,
            t_dp, a_s, pinchx, pinchy, dmg1, dmg2, beta, safety_factor)
        self.beam_section_name_tagbeam_section_name_tagbeam_section_name_tag = beam.name_tag
        self.col_section_name_tagcol_section_name_tagcol_section_name_tag = col.name_tag
        self.UpdateStoredDataUpdateStoredData()


class Skiadopoulos2021(MaterialModels):
    """
    Class that stores funcions and material properties of a steel double symmetric I-shape profile
        with Skiadopoulos 2021 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.
    The material model is valid only if the column is continuous.
    For more information about the empirical model for the computation of the parameters, see Skiadopoulos et Al. 2021.
    The vectors that forms the matrix used to compute the material model parameters (Kf_Ke_tests, Cw1_tests, Cf1_tests,
        Cw4_tests, Cf4_tests, Cw6_tests, Cf6_tests) are used as global throughout the class to optimise the program (given the fact that is constant everytime).

    @param MaterialModels: Parent abstract class.
    """
    global Kf_Ke_tests, Cw1_tests, Cf1_tests, Cw4_tests, Cf4_tests, Cw6_tests, Cf6_tests

    Kf_Ke_tests = [1.000, 0.153, 0.120, 0.090, 0.059, 0.031, 0.019, 0.009, 0.005, 0.004, 0.000]
    Kf_Ke_tests.reverse()
    Cw1_tests = [0.96, 0.96, 0.955, 0.94, 0.93, 0.90, 0.89, 0.89, 0.88, 0.88, 0.88]
    Cw1_tests.reverse()
    Cf1_tests = [0.035, 0.035, 0.033, 0.031, 0.018, 0.015, 0.013, 0.009, 0.009, 0.010, 0.010]
    Cf1_tests.reverse()
    Cw4_tests = [1.145, 1.145, 1.140, 1.133, 1.120, 1.115, 1.115, 1.11, 1.10, 1.10, 1.10]
    Cw4_tests.reverse()
    Cf4_tests = [0.145, 0.145, 0.123, 0.111, 0.069, 0.040, 0.040, 0.018, 0.010, 0.012, 0.012]
    Cf4_tests.reverse()
    Cw6_tests = [1.205, 1.2050, 1.2000, 1.1925, 1.1740, 1.1730, 1.1720, 1.1690, 1.1670, 1.1650, 1.1650]
    Cw6_tests.reverse()
    Cf6_tests = [0.165, 0.1650, 0.1400, 0.1275, 0.0800, 0.0500, 0.0500, 0.0180, 0.0140, 0.0120, 0.0120]
    Cf6_tests.reverse()

    def __init__(self, ID: int, d_c, bf_c, tf_c, I_c, d_b, tf_b, Fy, E, t_p,
        t_dp = 0.0, a_s = 0.03, pinchx = 0.25, pinchy = 0.75, dmg1 = 0.0, dmg2 = 0.0, beta = 0.0, safety_factor = False, t_fbp = 0):
```

```
        """
        Constructor of the class.

        @param ID (int): Unique material model ID.
        @param d_c (float): Column depth.
        @param bf_c (float): Column flange width.
        @param tf_c (float): Column flange thickness.
        @param I_c (float): Column moment of inertia (strong axis).
        @param d_b (float): Beam depth.
        @param tf_b (float): Beam flange thickness.
        @param Fy (float): Yield strength (if assume continous column, Fy of the web).
        @param E (float): Young modulus.
        @param t_p (float): Panel zone thickness.
        @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
        @param a_s (float, optional): Strain hardening. Defaults to 0.03.
        @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25
        @param pinchy (float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75
        @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
        @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
        @param beta (float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
        @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.
        @param t_fbp (float, optional): Thickness of the face bearing plate (if present). Defaults to 0.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: d_c needs to be positive.
        @exception NegativeValue: bf_c needs to be positive.
        @exception NegativeValue: tf_c needs to be positive.
        @exception NegativeValue: d_b needs to be positive.
        @exception NegativeValue: tf_b needs to be positive.
        @exception NegativeValue: Fy needs to be positive.
        @exception NegativeValue: E needs to be positive.
        @exception NegativeValue: t_p needs to be positive.
        @exception NegativeValue: a_s needs to be positive.
        """
        # Check
        if ID < 1: raise NegativeValue()
        if d_c < 0: raise NegativeValue()
        if bf_c < 0: raise NegativeValue()
        if tf_c < 0: raise NegativeValue()
        if d_b < 0: raise NegativeValue()
        if tf_b < 0: raise NegativeValue()
        if Fy < 0: raise NegativeValue()
        if E < 0: raise NegativeValue()
        if t_p < 0: raise NegativeValue()
        if a_s < 0: raise NegativeValue()
        if t_fbp < 0: raise NegativeValue()

        # Arguments
        self.IDID = ID
        self.d_cd_c = d_c
        self.bf_cbf_c = bf_c
        self.tf_ctf_c = tf_c
        self.I_cI_c = I_c
        self.d_bd_b = d_b
        self.tf_btf_b = tf_b
        self.FyFy = Fy
        self.EE = E
        self.t_pt_p = t_p
        self.t_dpt_dp = t_dp
        self.a_sa_s = a_s
        self.pinchxpinchx = pinchx
        self.pinchypinchy = pinchy
        self.dmg1dmg1 = dmg1
        self.dmg2dmg2 = dmg2
        self.betabeta = beta
        if safety_factor:
            self.RyRy = 1.2
        else:
            self.RyRy = 1.0
        self.t_fbpt_fbp = t_fbp

        # Initialized the parameters that are dependent from others
        self.beam_section_name_tagbeam_section_name_tag = "None"
        self.col_section_name_tagcol_section_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit()


    # Methods
    def ReInit(self):
        """
        Implementation of the homonym abstract method.
```

```
        See parent class DataManagement for detailed information.
        """
        # Check applicability
        self.CheckApplicabilityCheckApplicabilityCheckApplicability()

        # Memebers
        if self.beam_section_name_tagbeam_section_name_tag != "None": self.beam_section_name_tagbeam_section_name_tag = self.beam_section_name_tagbeam_section_name_tag + " (modified)"
        if self.col_section_name_tagcol_section_name_tag != "None": self.col_section_name_tagcol_section_name_tag = self.col_section_name_tagcol_section_name_tag + " (modified)"
        self.t_pzt_pz = self.t_pt_p + self.t_dpt_dp
        self.GG = self.EE/(2.0 * (1.0 + 0.30)) # Shear Modulus

        # Refined computation of the parameters for the backbone curve for the panel zone spring (Skiadopoulos et al. (2021))
        # Panel Zone Elastic Stiffness
        self.KsKs = self.t_pzt_pz*(self.d_cd_c-self.tf_ctf_c)*self.GG
        self.KbKb = 12.0*self.EE*(self.I_cI_c+self.t_dpt_dp*(self.d_cd_c-2.0*self.tf_ctf_c)**3/12.0)/(self.d_bd_b-0)**2
        self.KeKe = self.KsKs*self.KbKb/(self.KsKs+self.KbKb)

        # Column Flange Stiffness
        self.KsfKsf = 2.0*((self.tf_ctf_c+self.t_fbpt_fbp)*self.bf_cbf_c*self.GG)
        self.KbfKbf = 2.0*(12.0*self.EE*self.bf_cbf_c*(self.tf_ctf_c**3+self.t_fbpt_fbp**3)/12.0/(self.d_bd_b-0)**2)
        self.KfKf = self.KsfKsf*self.KbfKbf/(self.KsfKsf+self.KbfKbf)

        # Kf/Ke Calculation for Panel Zone Categorization
        self.Kf_KeKf_Ke = self.KfKf/self.KeKe

        # Panel Zone Strength Coefficients (results from tests for a_w_eff and a_f_eff)
        self.Cw1Cw1 = np.interp(self.Kf_KeKf_Ke, Kf_Ke_tests, Cw1_tests)
        self.Cf1Cf1 = np.interp(self.Kf_KeKf_Ke, Kf_Ke_tests, Cf1_tests)
        self.Cw4Cw4 = np.interp(self.Kf_KeKf_Ke, Kf_Ke_tests, Cw4_tests)
        self.Cf4Cf4 = np.interp(self.Kf_KeKf_Ke, Kf_Ke_tests, Cf4_tests)
        self.Cw6Cw6 = np.interp(self.Kf_KeKf_Ke, Kf_Ke_tests, Cw6_tests)
        self.Cf6Cf6 = np.interp(self.Kf_KeKf_Ke, Kf_Ke_tests, Cf6_tests)

        # Panel Zone Model
        self.V1V1 = self.FyFy*self.RyRy/math.sqrt(3)*(self.Cw1Cw1*(self.d_cd_c-self.tf_ctf_c)*self.t_pzt_pz + self.Cf1Cf1*2*(self.bf_cbf_c-self.t_pt_p)*self.tf_ctf_c)
        self.V4V4 = self.FyFy*self.RyRy/math.sqrt(3)*(self.Cw4Cw4*(self.d_cd_c-self.tf_ctf_c)*self.t_pzt_pz + self.Cf4Cf4*2*(self.bf_cbf_c-self.t_pt_p)*self.tf_ctf_c)
        self.V6V6 = self.FyFy*self.RyRy/math.sqrt(3)*(self.Cw6Cw6*(self.d_cd_c-self.tf_ctf_c)*self.t_pzt_pz + self.Cf6Cf6*2*(self.bf_cbf_c-self.t_pt_p)*self.tf_ctf_c)

        self.M1M1 = self.V1V1*(self.d_bd_b-self.tf_btf_b)
        self.M4M4 = self.V4V4*(self.d_bd_b-self.tf_btf_b)
        self.M6M6 = self.V6V6*(self.d_bd_b-self.tf_btf_b)

        self.Gamma_1Gamma_1 = self.V1V1/self.KeKe
        self.Gamma_4Gamma_4 = 4*self.Gamma_1Gamma_1
        self.Gamma_6Gamma_6 = 6*self.Gamma_1Gamma_1

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "Skiadopoulos2021"], # Tag for differentiating different data
            ["ID", self.IDID],
            ["beam_section_name_tag", self.beam_section_name_tagbeam_section_name_tag],
            ["col_section_name_tag", self.col_section_name_tagcol_section_name_tag],
            ["d_c", self.d_cd_c],
            ["bf_c", self.bf_cbf_c],
            ["tf_c", self.tf_ctf_c],
            ["I_c", self.I_cI_c],
            ["d_b", self.d_bd_b],
            ["tf_b", self.tf_btf_b],
            ["Fy", self.FyFy],
            ["E", self.EE],
            ["G", self.GG],
            ["t_p", self.t_pt_p],
            ["t_dp", self.t_dpt_dp],
            ["t_pz", self.t_pzt_pz],
            ["a_s", self.a_sa_s],
            ["pinchx", self.pinchxpinchx],
            ["pinchy", self.pinchypinchy],
```

```
            ["dmg1", self.dmg1dmg1],
            ["dmg2", self.dmg2dmg2],
            ["beta", self.betabeta],
            ["Ry", self.RyRy],
            ["Ks", self.KsKs],
            ["Kb", self.KbKb],
            ["Ke", self.KeKe],
            ["Ksf", self.KsfKsf],
            ["Kbf", self.KbfKbf],
            ["Kf", self.KfKf],
            ["Kf_Ke", self.Kf_KeKf_Ke],
            ["V1", self.V1V1],
            ["V4", self.V4V4],
            ["V6", self.V6V6],
            ["M1", self.M1M1],
            ["M4", self.M4M4],
            ["M6", self.M6M6],
            ["Gamma_1", self.Gamma_1Gamma_1],
            ["Gamma_4", self.Gamma_4Gamma_4],
            ["Gamma_6", self.Gamma_6Gamma_6],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self, plot = False, block = False):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        """
        print("")
        print("Requested info for Skiadopoulos 2021 material model Parameters, ID = {}".format(self.IDID))
        print("Sections associated, column: {} ".format(self.col_section_name_tagcol_section_name_tag))
        print("Sections associated, beam: {} ".format(self.beam_section_name_tagbeam_section_name_tag))
        print("Gamma_1 = {} rad".format(self.Gamma_1Gamma_1))
        print("Gamma_4 = {} rad".format(self.Gamma_4Gamma_4))
        print("Gamma_6 = {} rad".format(self.Gamma_6Gamma_6))
        print("M1 = {} kNm".format(self.M1M1/kNm_unit))
        print("M4 = {} kNm".format(self.M4M4/kNm_unit))
        print("M6 = {} kNm".format(self.M6M6/kNm_unit))
        print("")

        if plot:
            # Data for plotting
            x_axis = np.array([0.0, self.Gamma_1Gamma_1, self.Gamma_4Gamma_4, self.Gamma_6Gamma_6])
            y_axis = np.array([0.0, self.M1M1, self.M4M4, self.M6M6])/kNm_unit

            fig, ax = plt.subplots()
            ax.plot(x_axis, y_axis, 'k-')

            ax.set(xlabel='Rotation [rad]', ylabel='Moment [kNm]',
                title='Skiadopoulos 2021 material model (ID={})'.format(self.IDID))
            ax.grid()

            if block:
                plt.show()


    def CheckApplicability(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        Check = True
        # No checks
        if not Check:
            print("The validity of the equations is not fullfilled.")
            print("!!!!!!! WARNING !!!!!!! Check material model of Skiadopoulos 2021, ID=", self.IDID)
            print("")


    def Hysteretic(self):
        """
        Generate the material model Hysteretic (Skiadopoulos 2021) using the computed parameters.
        See _Hysteretic function for more information.
        """
        _Hysteretic(self.IDID, self.M1M1, self.Gamma_1Gamma_1, self.M4M4, self.Gamma_4Gamma_4, self.M6M6, self.Gamma_6Gamma_6,
            self.pinchxpinchx, self.pinchypinchy, self.dmg1dmg1, self.dmg2dmg2, self.betabeta)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()
```

```


class Skiadopoulos2021SteelIShape(Skiadopoulos2021):
    """
    Class that is the children of Skiadopoulos2021 and combine the class SteelIShape (section) to retrieve the information needed.

    @param Skiadopoulos2021: Parent class.
    """
    def __init__(self, ID: int, col: SteelIShape, beam: SteelIShape,
        t_dp=0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0, dmg2=0, beta=0, safety_factor=False, t_fbp = 0):
        """
        Constructor of the class. It passes the arguments into the parent class to generate the combination of the parent class
            and the section class SteelIShape.
        The copy of the sections (col and beam) passed are stored in the member variable self.colcol and self.beambeam.

        @param ID (int): Unique material model ID.
        @param col (SteelIShape): SteelIShape column section object.
        @param beam (SteelIShape): SteelIShape beam section object.
        @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
        @param a_s (float, optional): Strain hardening. Defaults to 0.03.
        @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25.
        @param pinchy (float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75.
        @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
        @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
        @param beta (float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
        @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.
        @param t_fbp (float, optional): Thickness of the face bearing plate (if present). Defaults to 0.
        """
        self.colcol = deepcopy(col)
        self.beambeam = deepcopy(beam)
        super().__init__(ID, col.d, col.bf, col.tf, col.Iy, beam.d, beam.tf, col.Fy_web, col.E, col.tw,
            t_dp=t_dp, a_s=a_s, pinchx=pinchx, pinchy=pinchy, dmg1=dmg1, dmg2=dmg2, beta=beta, safety_factor=safety_factor, t_fbp=t_fbp)
        self.beam_section_name_tagbeam_section_name_tagbeam_section_name_tag = beam.name_tag
        self.col_section_name_tagcol_section_name_tagcol_section_name_tag = col.name_tag
        self.UpdateStoredDataUpdateStoredData()


class Skiadopoulos2021RCS(Skiadopoulos2021):
    """
    WIP: Class that is the children of Skiadopoulos2021 and it's used for the panel zone spring in a RCS (RC column continous, Steel beam).

    @param Skiadopoulos2021: Parent class.
    """
    def __init__(self, ID: int, beam: SteelIShape, d_col, t_fbp = 0,
        t_dp=0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0, dmg2=0, beta=0, safety_factor=False):
        """
        Constructor of the class. It passes the arguments into the parent class to generate the combination of the parent class
            and the section class SteelIShape.
        The copy of the section (beam) passed is stored in the member variable self.beambeam.

        @param ID (int): Unique material model ID.
        @param beam (SteelIShape): SteelIShape beam section object.
        @param d_col (float): Depth of the RC column (continous)
        @param t_fbp (float, optional): Thickness of the face bearing plate (if present). Defaults to 0.
        @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
        @param a_s (float, optional): Strain hardening. Defaults to 0.03.
        @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25
        @param pinchy (float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75
        @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
        @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
        @param beta (float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
        @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.
        """
        self.beambeam = deepcopy(beam)
        super().__init__(ID, beam.d, beam.bf, beam.tf, beam.Iy, d_col, 0, beam.Fy_web, beam.E, beam.tw,
            t_dp=t_dp, a_s=a_s, pinchx=pinchx, pinchy=pinchy, dmg1=dmg1, dmg2=dmg2, beta=beta, safety_factor=safety_factor, t_fbp=t_fbp)
        self.beam_section_name_tagbeam_section_name_tagbeam_section_name_tag = beam.name_tag
```

```
        self.UpdateStoredDataUpdateStoredData()


class UnconfMander1988(MaterialModels):
    """
    Class that stores funcions and material properties of a RC rectangular or circular section
        with Mander 1988 as the material model for the unconfined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.
    For more information about the empirical model for the computation of the parameters, see Mander et Al. 1988, Karthik and Mander 2011 and SIA 262:2012.

    @param MaterialModels: Parent abstract class.
    """
    def __init__(self, ID: int, fc, Ec, ec = 1, ecp = 1, fct = -1, et = -1, beta = 0.1):
        """
        Constructor of the class.

        @param ID (int): Unique material model ID.
        @param fc (float): Compressive concrete yield strength (needs to be negative).
        @param Ec (float): Young modulus.
        @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
        @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
        @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param beta (float, optional): Loating point value defining the exponential curve parameter to define the residual stress.
            Defaults to 0.1 (according to OpenSeesPy documentation)

        @exception NegativeValue: ID needs to be a positive integer.
        @exception PositiveValue: fc needs to be negative.
        @exception NegativeValue: Ec needs to be positive.
        @exception PositiveValue: ec needs to be negative if different from 1.
        @exception PositiveValue: ecp needs to be positive if different from 1.
        @exception NegativeValue: fct needs to be positive if different from -1.
        @exception NegativeValue: et needs to be positive if different from -1.
        """
        # Check
        if ID < 0: raise NegativeValue()
        if fc > 0: raise PositiveValue()
        if Ec < 0: raise NegativeValue()
        if ec != 1 and ec > 0: raise PositiveValue()
        if ecp != 1 and ecp > 0: raise PositiveValue()
        if fct != -1 and fct < 0: raise NegativeValue()
        if et != -1 and et < 0: raise NegativeValue()

        # Arguments
        self.IDID = ID
        self.fcfc = fc
        self.EcEc = Ec
        self.betabeta = beta

        # Initialized the parameters that are dependent from others
        self.section_name_tagsection_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit(ec, ecp, fct, et)

    # Methods
    def ReInit(self, ec = 1, ecp = 1, fct = -1, et = -1):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
        @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
        @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        """
        # Check applicability
        self.CheckApplicabilityCheckApplicabilityCheckApplicability()

        # Arguments
        self.ecec = self.Compute_ecCompute_ec() if ec == 1 else ec
        self.ecpecp = self.Compute_ecpCompute_ecp() if ecp == 1 else ecp
        self.fctfct = self.Compute_fctCompute_fct() if fct == -1 else fct
        self.etet = self.Compute_etCompute_et() if et == -1 else et

        # Members
        self.ecuecu = self.Compute_ecuCompute_ecu()
        if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
```

```
self.section_name_tagsection_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "UnconfMander1988"], # Tag for differentiating different data
            ["ID", self.IDID],
            ["section_name_tag", self.section_name_tagsection_name_tag],
            ["fc", self.fcfc],
            ["Ec", self.EcEc],
            ["ec", self.ecec],
            ["ecp", self.ecpecp],
            ["ecu", self.ecuecu],
            ["fct", self.fctfct],
            ["et", self.etet],
            ["beta", self.betabeta],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self, plot = False, block = False, concrete04 = True):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        @param concrete04 (bool, optional): Option to show in the plot the concrete04 or concrete01 if False. Defaults to True.
        """
        print("")
        print("Requested info for Unconfined Mander 1988 material model Parameters, ID = {}".format(self.IDID))
        print("Section associated: {} ".format(self.section_name_tagsection_name_tag))
        print('Concrete strength fc = {} MPa'.format(self.fcfc/MPa_unit))
        print('Strain at maximal strength ec = {}'.format(self.ecec))
        print('Maximal strain ecu = {}'.format(self.ecuecu))
        print("")

        if plot:
            fig, ax = plt.subplots()
            if concrete04:
                PlotConcrete04(self.fcfc, self.EcEc, self.ecec, self.ecuecu, "U", ax, self.IDID)
            else:
                PlotConcrete01(self.fcfc, self.ecec, 0, self.ecuecu, ax, self.IDID)

            if block:
                plt.show()


    def CheckApplicability(self):
        """
        Implementation of the homonym abstract method.
        See parent class MaterialModels for detailed information.
        """
        Check = True
        if self.fcfc < -110*MPa_unit: # Deierlein 1999
            Check = False
            print("With High Strength concrete (< -110 MPa), a better material model should be used (see Abdesselam et Al. 2019")
        if not Check:
            print("The validity of the equations is not fullfilled.")
            print("!!!!!!! WARNING !!!!!!! Check material model of Unconfined Mander 1988, ID=", self.IDID)
            print("")


    def Compute_ec(self):
        """
        Method that computes the compressive concrete yield strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        # return -0.002 # Alternative: Mander et Al. 1988
        return -0.0015 + self.fcfc/MPa_unit/70000 # Karthik Mander 2011

    def Compute_ecp(self):
        """
        Method that computes the compressive concrete spalling strain.
```

```
        For more information, see Mander et Al. 1988.

        @returns float: Strain
        """
        return 2.0*self.ecec


    def Compute_fct(self):
        """
        Method that computes the tensile concrete yield stress.
        For more information, see SIA 262:2012.

        @returns float: Stress.
        """
        return 0.30 * math.pow(-self.fcfc/MPa_unit, 2/3) * MPa_unit


    def Compute_et(self):
        """
        Method that computes the tensile concrete yield strain.
        For more information, see Mander et Al. 1988 (eq 45).

        @returns float: Strain.
        """
        return self.fctfct/self.EcEc


    def Compute_ecu(self):
        """
        Method that computes the compressive concrete failure strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        # return -0.004 # Alternative: Mander et Al. 1988
        return -0.012 - 0.0001 * self.fcfc/MPa_unit # Karthik Mander 2011

    def Concrete01(self):
        """
        Generate the material model Concrete01 for unconfined concrete using the computed parameters.
        See _Concrete01 function for more information. Use this method or Concrete04, not both (only one material model for ID).
        """
        _Concrete01(self.IDID, self.ecec, self.fcfc, self.ecuecu)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


    def Concrete04(self):
        """
        Generate the material model Concrete04 for unconfined concrete (Mander 1988) using the computed parameters.
        See _Concrete04 function for more information. Use this method or Concrete01, not both (only one material model for ID).
        """
        _Concrete04(self.IDID, self.fcfc, self.ecec, self.ecuecu, self.EcEc, self.fctfct, self.etet, self.betabeta)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


class UnconfMander1988RCRectShape(UnconfMander1988):
    """
    Class that is the children of UnconfMander1988 and combine the class RCRectShape (section) to retrieve the information needed.

    @param UnconfMander1988: Parent class.
    """
    def __init__(self, ID: int, section: RCRectShape, ec=1, ecp=1, fct=-1, et=-1, beta=0.1):
        """
        Constructor of the class. It passes the arguments into the parent class to generate the combination of the parent class
            and the section class RCRectShape.
        The copy of the section passed is stored in the member variable self.sectionsection.

        @param ID (int): Unique material model ID.
        @param section (RCRectShape): RCRectShape section object.
        @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
        @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
        @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param beta (float, optional): Loating point value defining the exponential curve parameter to define the residual stress.
```

```
            Defaults to 0.1 (according to OpenSeesPy documentation)
        """
        self.sectionsection = deepcopy(section)
        super().__init__(ID, section.fc, section.Ec, ec=ec, ecp=ecp, fct=fct, et=et, beta=beta)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()


class UnconfMander1988RCCircShape(UnconfMander1988):
    """
    Class that is the children of UnconfMander1988 and combine the class RCCircShape (section) to retrieve the information needed.

    @param UnconfMander1988: Parent class.
    """
    def __init__(self, ID: int, section: RCCircShape, ec=1, ecp=1, fct=-1, et=-1, beta=0.1):
        """
        Constructor of the class. It passes the arguments into the parent class to generate the combination of the parent class
            and the section class RCCircShape.
        The copy of the section passed is stored in the member variable self.sectionsection.

        @param ID (int): Unique material model ID.
        @param section (RCCircShape): RCCircShape section object.
        @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
        @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
        @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param beta (float, optional): Loating point value defining the exponential curve parameter to define the residual stress.
            Defaults to 0.1 (according to OpenSeesPy documentation)
        """
        self.sectionsection = deepcopy(section)
        super().__init__(ID, section.fc, section.Ec, ec=ec, ecp=ecp, fct=fct, et=et, beta=beta)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()


class ConfMander1988Rect(MaterialModels):
    """
    Class that stores funcions and material properties of a RC rectangular section
        with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.
    For more information about the empirical model for the computation of the parameters, see Mander et Al. 1988, Karthik and Mander 2011 and SIA 262:2012.
    The array array_fl2 and curve curve_fl1 are the parameter of the digitized table used to extrapolate the confinement factor;
        they are used as global throughout the ConfMander1988Rect material model to optimise the program (given the fact that is constant everytime).

    @param MaterialModels: Parent abstract class.
    """
    global array_fl2, curve_fl1

    curve_fl1 = np.arange(0, 0.3+0.02, 0.02)
    array_fl2 = [None] * len(curve_fl1)

    array_fl2[0] = [[1.0, 0],
        [1.0026455026455026, 6.5359477124183E-4],
        [1.0423280423280423, 0.01699346405228758],
        [1.0846560846560847, 0.037254901960784306],
        [1.119047619047619, 0.05686274509803921],
        [1.1455026455026456, 0.0784313725490196],
        [1.1666666666666667, 0.09607843137254903],
        [1.193121693121693, 0.11830065359477124],
        [1.208994708994709, 0.1392156862745098],
        [1.2248677248677249, 0.15751633986928104],
        [1.2380952380952381, 0.17712418300653593],
        [1.2513227513227514, 0.19673202614379084],
        [1.2645502645502646, 0.21699346405228756],
        [1.2724867724867726, 0.23660130718954248],
        [1.2804232804232805, 0.2594771241830065],
        [1.2883597883597884, 0.2777777777777778],
        [1.2936507936507937, 0.3]]

    array_fl2[1] = [[1.1349206349206349, 0.01895424836601307],
        [1.1825396825396826, 0.03790849673202614],
        [1.2222222222222223, 0.05686274509803921],
        [1.2513227513227514, 0.07777777777777777],
        [1.2724867724867726, 0.09738562091503267],
        [1.291005291005291, 0.11895424836601308],
        [1.3174603174603174, 0.13856209150326795],
        [1.335978835978836, 0.1588235294117647],
```

```
        [1.3518518518518519, 0.17777777777777776],
        [1.3677248677248677, 0.19738562091503267],
        [1.3783068783068784, 0.2176470588235294],
        [1.3941798941798942, 0.238562091503268],
        [1.41005291005291, 0.2594771241830065],
        [1.4126984126984126, 0.28104575163398693],
        [1.4232804232804233, 0.3]]

    array_fl2[2] = [[1.246031746031746, 0.037254901960784306],
        [1.298941798941799, 0.05751633986928104],
        [1.335978835978836, 0.07712418300653595],
        [1.3650793650793651, 0.09869281045751634],
        [1.3888888888888888, 0.11699346405228757],
        [1.4153439153439153, 0.1392156862745098],
        [1.439153439153439, 0.1568627450980392],
        [1.4603174603174602, 0.17712418300653593],
        [1.4735449735449735, 0.1980392156862745],
        [1.4894179894179893, 0.21633986928104573],
        [1.5052910052910053, 0.23790849673202616],
        [1.5211640211640212, 0.2581699346405229],
        [1.5317460317460316, 0.2777777777777778],
        [1.5423280423280423, 0.3]]

    array_fl2[3] = [[1.3544973544973544, 0.05686274509803921],
        [1.3994708994708995, 0.07777777777777777],
        [1.4417989417989419, 0.09738562091503267],
        [1.4735449735449735, 0.11699346405228757],
        [1.4947089947089947, 0.13790849673202613],
        [1.5238095238095237, 0.15751633986928104],
        [1.5423280423280423, 0.17777777777777776],
        [1.560846560846561, 0.1980392156862745],
        [1.5820105820105819, 0.21633986928104573],
        [1.5952380952380953, 0.2372549019607843],
        [1.6137566137566137, 0.2588235294117647],
        [1.626984126984127, 0.27908496732026145],
        [1.6375661375661377, 0.3]]

    array_fl2[4] = [[1.455026455026455, 0.07647058823529412],
        [1.5, 0.09738562091503267],
        [1.5423280423280423, 0.1196078431372549],
        [1.574074074074074, 0.13856209150326795],
        [1.6058201058201058, 0.1588235294117647],
        [1.6296296296296298, 0.17777777777777776],
        [1.6534391534391535, 0.1980392156862745],
        [1.671957671957672, 0.21699346405228756],
        [1.6904761904761905, 0.238562091503268],
        [1.7063492063492065, 0.2588235294117647],
        [1.716931216931217, 0.27908496732026145],
        [1.7248677248677249, 0.3]]

    array_fl2[5] = [[1.5634920634920635, 0.09869281045751634],
        [1.6058201058201058, 0.11895424836601308],
        [1.6428571428571428, 0.13790849673202613],
        [1.6746031746031746, 0.1588235294117647],
        [1.701058201058201, 0.17908496732026144],
        [1.7222222222222223, 0.19673202614379084],
        [1.7433862433862433, 0.2176470588235294],
        [1.7698412698412698, 0.2392156862745098],
        [1.783068783068783, 0.2581699346405229],
        [1.798941798941799, 0.27908496732026145],
        [1.8095238095238095, 0.3]]

    array_fl2[6] = [[1.6507936507936507, 0.11633986928104575],
        [1.693121693121693, 0.13856209150326795],
        [1.7328042328042328, 0.15751633986928104],
        [1.7645502645502646, 0.17843137254901958],
        [1.7910052910052912, 0.1980392156862745],
        [1.8148148148148149, 0.2176470588235294],
        [1.8333333333333335, 0.23790849673202616],
        [1.8571428571428572, 0.2581699346405229],
        [1.8677248677248677, 0.2797385620915033],
        [1.8915343915343916, 0.3]]

    array_fl2[7] = [[1.753968253968254, 0.14052287581699346],
        [1.7883597883597884, 0.15816993464052287],
        [1.8174603174603174, 0.17843137254901958],
        [1.8412698412698414, 0.19738562091503267],
        [1.8677248677248677, 0.21699346405228756],
        [1.8835978835978837, 0.2372549019607843],
        [1.9047619047619047, 0.257516339869281],
        [1.925925925925926, 0.27908496732026145],
        [1.9417989417989419, 0.3]]

    array_fl2[8] = [[1.8386243386243386, 0.16013071895424835],
        [1.8703703703703702, 0.17908496732026144],
        [1.8994708994708995, 0.1980392156862745],
        [1.925925925925926, 0.2176470588235294],
```

```
        [1.9497354497354498, 0.23790849673202616],
        [1.9761904761904763, 0.2588235294117647],
        [1.992063492063492, 0.27908496732026145],
        [2.0132275132275135, 0.3]]

    array_fl2[9] = [[1.9179894179894181, 0.1823529411764706],
        [1.939153439153439, 0.19869281045751636],
        [1.9682539682539684, 0.21895424836601307],
        [1.992063492063492, 0.238562091503268],
        [2.0132275132275135, 0.2568627450980392],
        [2.0396825396825395, 0.27908496732026145],
        [2.060846560846561, 0.3]]

    array_fl2[10] = [[1.9761904761904763, 0.19673202614379084],
        [2.007936507936508, 0.21633986928104573],
        [2.0343915343915344, 0.2372549019607843],
        [2.066137566137566, 0.257516339869281],
        [2.08994708994709, 0.2784313725490196],
        [2.111111111111111, 0.3]]

    array_fl2[11] = [[2.044973544973545, 0.21633986928104573],
        [2.0767195767195767, 0.238562091503268],
        [2.1084656084656084, 0.2581699346405229],
        [2.134920634920635, 0.28039215686274505],
        [2.158730158730159, 0.3]]

    array_fl2[12] = [[2.113756613756614, 0.2372549019607843],
        [2.1455026455026456, 0.257516339869281],
        [2.1719576719576716, 0.2797385620915033],
        [2.193121693121693, 0.3]]

    array_fl2[13] = [[2.177248677248677, 0.2581699346405229],
        [2.2063492063492065, 0.27908496732026145],
        [2.2275132275132274, 0.3]]

    array_fl2[14] = [[2.2407407407407405, 0.2784313725490196],
        [2.261904761904762, 0.3]]

    array_fl2[15] = [[2.2962962962962963, 0.3]]

    def __init__(self, ID: int, bc, dc, Ac, fc, Ec, nr_bars, D_bars, wx_top: np.ndarray, wx_bottom: np.ndarray, wy: np.ndarray, s, D_hoops, rho_s_x, rho_s_y, fs,
        ec = 1, ecp = 1, fct = -1, et = -1, esu = -1, beta = 0.1):
        """
        Constructor of the class.

        @param ID (int): Unique material model ID.
        @param bc (float): Width of the confined core (from the centerline of the hoops, according to Mander et Al. 1988).
        @param dc (float): Depth of the confined core (from the centerline of the hoops, according to Mander et Al. 1988).
        @param Ac (float): Area of the confined core (according to Mander et Al. 1988).
        @param fc (float): Compressive concrete yield strength (needs to be negative).
        @param Ec (float): Young modulus.
        @param nr_bars (float): Number of reinforcement (allow float for computing the equivalent nr_bars with different reinforcement areas).
        @param D_bars (float): Diameter of the vertical reinforcing bars.
        @param wx_top (np.ndarray): Vector of 1 dimension that defines the distance between top vertical bars in x direction (NOT CENTERLINE DISTANCES).
        @param wx_bottom (np.ndarray): Vector of 1 dimension that defines the distance between bottom vertical bars in x direction (NOT CENTERLINE DISTANCES).
        @param wy (np.ndarray): Vector of 1 dimension that defines the distance between vertical bars in y direction (lateral) (NOT CENTERLINE DISTANCES).
        @param s (float): Vertical spacing between hoops.
        @param D_hoops (float): Diameter of hoops.
        @param rho_s_x (float): Ratio of the transversal area of the hoops to the associated concrete area in the x direction.
        @param rho_s_y (float): Ratio of the transversal area of the hoops to the associated concrete area in the y direction.
        @param fs (float): Yield stress for the hoops.
        @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
        @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
        @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
        @param beta (float, optional): Loating point value defining the exponential curve parameter to define the residual stress.
            Defaults to 0.1 (according to OpenSeesPy documentation)

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: bc needs to be positive.
        @exception NegativeValue: dc needs to be positive.
```

```
        @exception NegativeValue: Ac needs to be positive.
        @exception PositiveValue: fc needs to be negative.
        @exception NegativeValue: Ec needs to be positive.
        @exception NegativeValue: nr_bars needs to be positive.
        @exception NegativeValue: D_bars needs to be positive.
        @exception NegativeValue: s needs to be positive.
        @exception NegativeValue: D_hoops needs to be positive.
        @exception NegativeValue: rho_s_x needs to be positive.
        @exception NegativeValue: rho_s_y needs to be positive.
        @exception NegativeValue: fs needs to be positive.
        @exception PositiveValue: ec needs to be negative if different from 1.
        @exception PositiveValue: ecp needs to be negative if different from 1.
        @exception NegativeValue: fct needs to be positive if different from -1.
        @exception NegativeValue: et needs to be positive if different from -1.
        @exception NegativeValue: esu needs to be positive if different from -1.
        """
        # Check
        if ID < 1: raise NegativeValue()
        if bc < 0: raise NegativeValue()
        if dc < 0: raise NegativeValue()
        if Ac < 0: raise NegativeValue()
        if fc > 0: raise PositiveValue()
        if Ec < 0: raise NegativeValue()
        if nr_bars < 0: raise NegativeValue()
        if D_bars < 0: raise NegativeValue()
        if s < 0: raise NegativeValue()
        if D_hoops < 0: raise NegativeValue()
        if rho_s_x < 0: raise NegativeValue()
        if rho_s_y < 0: raise NegativeValue()
        if fs < 0: raise NegativeValue()
        if ec != 1 and ec > 0: raise PositiveValue()
        if ecp != 1 and ecp > 0: raise PositiveValue()
        if fct != -1 and fct < 0: raise NegativeValue()
        if et != -1 and et < 0: raise NegativeValue()
        if esu != -1 and esu < 0: raise NegativeValue()

        # Arguments
        self.IDID = ID
        self.bcbc = bc
        self.dcdc = dc
        self.AcAc = Ac
        self.fcfc = fc
        self.EcEc = Ec
        self.nr_barsnr_bars = nr_bars
        self.D_barsD_bars = D_bars
        self.wx_topwx_top = copy(wx_top)
        self.wx_bottomwx_bottom = copy(wx_bottom)
        self.wywy = copy(wy)
        self.ss = s
        self.D_hoopsD_hoops = D_hoops
        self.rho_s_xrho_s_x = rho_s_x
        self.rho_s_yrho_s_y = rho_s_y
        self.fsfs = fs
        self.esuesu = 0.05 if esu == -1 else esu # Mander 1988
        self.betabeta = beta

        # Initialized the parameters that are dependent from others
        self.section_name_tagsection_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit(ec, ecp, fct, et)

    def ReInit(self, ec = 1, ecp = 1, fct = -1, et = -1):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
    according to Karthik and Mander 2011.
        @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
    to Mander 1988.
        @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
    according to SIA 262:2012.
        @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
    according to SIA 262:2012.
        """
        # Check applicability
        self.CheckApplicabilityCheckApplicabilityCheckApplicability()

        # Arguments
        self.ecec = self.Compute_ecCompute_ec() if ec == 1 else ec
        self.ecpecp = self.Compute_ecpCompute_ecp() if ecp == 1 else ecp
        self.fctfct = self.Compute_fctCompute_fct() if fct == -1 else fct
        self.etet = self.Compute_etCompute_et() if et == -1 else et

        # Members (according to Mander 1988, confined concrete)
        self.ecuecu = self.Compute_ecuCompute_ecu()
        self.AiAi = self.ComputeAiComputeAi()
```

```
        self.AeAe = (self.AcAc - self.AiAi) * (1.0 - (self.ss-self.D_hoopsD_hoops)/2.0/self.bcbc)*(1.0 - (self.ss-self.D_hoopsD_hoops)/2.0/self.dcdc)
        self.rho_ccrho_cc = self.nr_barsnr_bars*self.D_barsD_bars**2/4.0*math.pi / self.AcAc
        self.AccAcc = self.AcAc*(1.0-self.rho_ccrho_cc)
        self.keke = self.AeAe/self.AccAcc
        self.fl_xfl_x = -self.rho_s_xrho_s_x * self.fsfs
        self.fl_yfl_y = -self.rho_s_yrho_s_y * self.fsfs
        self.K_comboK_combo = self.ComputeConfinementFactorComputeConfinementFactor()
        self.fccfcc = self.fcfc * self.K_comboK_combo
        self.eccecc = self.Compute_eccCompute_ecc()
        self.eccueccu = self.Compute_eccuCompute_eccu()
        if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag = self.section_name_tagsection_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    # Methods
    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "ConfMander1988rect"], # Tag for differentiating different data
            ["ID", self.IDID],
            ["section_name_tag", self.section_name_tagsection_name_tag],
            ["bc", self.bcbc],
            ["dc", self.dcdc],
            ["Ac", self.AcAc],
            ["fc", self.fcfc],
            ["Ec", self.EcEc],
            ["ec", self.ecec],
            ["ecp", self.ecpecp],
            ["ecu", self.ecuecu],
            ["fct", self.fctfct],
            ["et", self.etet],
            ["fcc", self.fccfcc],
            ["ecc", self.eccecc],
            ["eccu", self.eccueccu],
            ["beta", self.betabeta],
            ["nr_bars", self.nr_barsnr_bars],
            ["D_bars", self.D_barsD_bars],
            ["wx_top", self.wx_topwx_top],
            ["wx_bottom", self.wx_bottomwx_bottom],
            ["wy", self.wywy],
            ["s", self.ss],
            ["D_hoops", self.D_hoopsD_hoops],
            ["rho_s_x", self.rho_s_xrho_s_x],
            ["rho_s_y", self.rho_s_yrho_s_y],
            ["fs", self.fsfs],
            ["esu", self.esuesu],
            ["Ai", self.AiAi],
            ["Ae", self.AeAe],
            ["rho_cc", self.rho_ccrho_cc],
            ["Acc", self.AccAcc],
            ["ke", self.keke],
            ["fl_x", self.fl_xfl_x],
            ["fl_y", self.fl_yfl_y],
            ["K_combo", self.K_comboK_combo],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self, plot = False, block = False, concrete04 = True):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        @param concrete04 (bool, optional): Option to show in the plot the concrete04 or concrete01 if False. Defaults to True.
        """
        print("")
        print("Requested info for Confined Mander 1988 (rectangular) material model Parameters, ID = {}".format(self.IDID))
        print("Section associated: {} ".format(self.section_name_tagsection_name_tag))
        print('Concrete strength fc = {} MPa'.format(self.fcfc/MPa_unit))
        print('Concrete strength confined fcc = {} MPa'.format(self.fccfcc/MPa_unit))
        print('Strain at maximal strength ec = {}'.format(self.ecec))
        print('Strain at maximal strength confined ecc = {}'.format(self.eccecc))
        print('Maximal strain ecu = {}'.format(self.ecuecu))
        print('Maximal strain confined eccu = {}'.format(self.eccueccu))
        print("")

```

```
        if plot:
            fig, ax = plt.subplots()
            if concrete04:
                PlotConcrete04(self.fccfcc, self.EcEc, self.eccecc, self.eccueccu, "C", ax, self.IDID)
            else:
                PlotConcrete01(self.fccfcc, self.eccecc, 0.0, self.eccueccu, ax, self.IDID)

            if block:
                plt.show()


    def CheckApplicability(self):
        """
        Implementation of the homonym abstract method.
        See parent class MaterialModels for detailed information.
        """
        Check = True
        if self.fcfc < -110*MPa_unit: # Deierlein 1999
            Check = False
            print("With High Strength concrete (< -110 MPa), a better material model should be used (see Abdesselam et Al. 2019")
        if not Check:
            print("The validity of the equations is not fullfilled.")
            print("!!!!!!! WARNING !!!!!!! Check material model of Confined Mander 1988, ID=", self.IDID)
            print("")


    def Compute_ec(self):
        """
        Method that computes the compressive concrete yield strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        # return -0.002 # Alternative: Mander et Al. 1988
        return -0.0015 + self.fcfc/MPa_unit/70000 # Karthik Mander 2011


    def Compute_ecp(self):
        """
        Method that computes the compressive concrete spalling strain.
        For more information, see Mander et Al. 1988.

        @returns float: Strain
        """
        return 2.0*self.ecec


    def Compute_fct(self):
        """
        Method that computes the tensile concrete yield stress.
        For more information, see SIA 262:2012. Assume that the confinement do not play an essential role in tension.

        @returns float: Stress.
        """
        return 0.30 * math.pow(-self.fcfc/MPa_unit, 2/3) * MPa_unit


    def Compute_et(self):
        """
        Method that computes the tensile concrete yield strain.
        For more information, see Mander et Al. 1988 (eq 45).

        @returns float: Strain.
        """
        return self.fctfct/self.EcEc


    def Compute_ecu(self):
        """
        Method that computes the compressive concrete failure strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        # return -0.004 # Alternative: Mander et Al. 1988
        return -0.012 - 0.0001 * self.fcfc/MPa_unit # Karthik Mander 2011


    def Compute_ecc(self):
        """
        Method that computes the compressive confined concrete yield strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
```

```
        """
        return (1.0 + 5.0 * (self.K_comboK_combo-1.0)) * self.ecec # Karthik Mander 2011


    def Compute_eccu(self):
        """
        Method that computes the compressive confined concrete failure strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        # return -0.004 + (1.4*(self.rho_s_x+self.rho_s_y)*self.esu*self.fs) / self.fcc # Alternative: Prof. Katrin Beyer
        return 5*self.eccecc # Karthik Mander 2011


    def ComputeAi(self):
        """
        Method that computes the ineffectual area.
        For more information, see Mander et Al. 1988.

        @returns float: Area.
        """
        return ( np.sum(np.multiply(self.wywy, self.wywy))*2.0 +
            np.sum(np.multiply(self.wx_topwx_top, self.wx_topwx_top)) +
            np.sum(np.multiply(self.wx_bottomwx_bottom, self.wx_bottomwx_bottom)) ) / 6.0


    def ComputeConfinementFactor(self):
        """
        Method that computes the confinement factor using the digitized table from Mander et Al. 1988 that
            extrapolates the factor using the lateral confining stress in the two direction.

        @exception NoApplicability: The table from Mander accept ratio of fl/fc smaller than 0.3.
        @exception NoApplicability: The table from Mander accept ratio of fl/fc smaller than 0.3.
        @exception NegativeValue: fl1_ratio needs to be positive.
        @exception NegativeValue: fl2_ratio needs to be positive.

        @returns float: Confinement factor.
        """
        if self.fl_xfl_x == self.fl_yfl_y:
            return -1.254 + 2.254 * math.sqrt(1.0+7.94*self.fl_xfl_x*self.keke/self.fcfc) - 2.0*self.fl_xfl_x*self.keke/self.fcfc # in Mander, it has a prime
        else:
            fl2_ratio = max(self.fl_xfl_x*self.keke/self.fcfc, self.fl_yfl_y*self.keke/self.fcfc)
            fl1_ratio = min(self.fl_xfl_x*self.keke/self.fcfc, self.fl_yfl_y*self.keke/self.fcfc)

            if fl1_ratio > 0.3: raise NoApplicability()
            if fl2_ratio > 0.3: raise NoApplicability()
            if fl1_ratio < 0: raise NegativeValue()
            if fl2_ratio < 0: raise NegativeValue()

            # choose one or two curves
            for ii, fl1 in enumerate(curve_fl1):
                if fl1 == fl1_ratio:
                    # one curve
                    # choose curve
                    # curve_fl2 = [curve for ii, curve in enumerate(array_fl2) if index[ii]][0]
                    curve_fl2 = array_fl2[ii]

                    # Take value (interpole)
                    K = [item[0] for item in curve_fl2]
                    fl2 = [item[1] for item in curve_fl2]
                    K_res =  np.interp(fl2_ratio, fl2, K)

                    #TODO: to check fucntion:
                    # fig, ax = plt.subplots()
                    # ax.plot(fl2, K, 'k-')
                    # ax.scatter(fl2_ratio, K_res, color='k')
                    # ax.grid()
                    # plt.show()
                    return K_res

                # two curves
                if fl1 > fl1_ratio:
                    fl1_max = fl1
                    fl1_min = curve_fl1[ii-1]
                    curve_fl2_max = array_fl2[ii]
                    curve_fl2_min = array_fl2[ii-1]

                    # Take the values (interpole)
                    K_max = [item[0] for item in curve_fl2_max]
                    fl2_max = [item[1] for item in curve_fl2_max]
                    K_res_max =  np.interp(fl2_ratio, fl2_max, K_max)

                    K_min = [item[0] for item in curve_fl2_min]
```

```
                    fl2_min = [item[1] for item in curve_fl2_min]
                    K_res_min =  np.interp(fl2_ratio, fl2_min, K_min)

                    # interpole with distance from fl1 for fl2
                    # should be logarithmic interpolation but error negligibile
                    K_res = np.interp(fl1_ratio, [fl1_min, fl1_max], [K_res_min, K_res_max])
                    return K_res


    def Concrete01(self):
        """
        Generate the material model Concrete01 for rectangular section confined concrete (Mander 1988).
        See _Concrete01 function for more information. Use this method or Concrete04, not both (only one material model for ID).
        """
        _Concrete01(self.IDID, self.ecc ecc, self.fccfcc, self.eccueccu)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


    def Concrete04(self):
        """
        Generate the material model Concrete04 for rectangular section confined concrete (Mander 1988).
        See _Concrete04 function for more information. Use this method or Concrete01, not both (only one material model for ID).
        """
        _Concrete04(self.IDID, self.fccfcc, self.eccecc, self.eccueccu, self.EcEc, self.fctfct, self.etet, self.betabeta)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


class ConfMander1988RectRCRectShape(ConfMander1988Rect):
    """
    Class that is the children of ConfMander1988Rect and combine the class RCRectShape (section) to retrieve the information needed.

    @param ConfMander1988Rect: Parent class.
    """
    def __init__(self, ID: int, section: RCRectShape, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1):
        """
        Constructor of the class. It passes the arguments into the parent class to generate the combination of the parent class
            and the section class RCRectShape. wx_bottom, wx_top and wy are computed using the private method __Compute_w that
            and the member variable bars_ranges_position_y and bars_position_x from the section passed.
        The copy of the section passed is stored in the member variable self.sectionsection.

        @param ID (int): Unique material model ID.
        @param section (RCRectShape): RCRectShape section object.
        @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
        @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
        @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
        @param beta (float, optional): Loating point value defining the exponential curve parameter to define the residual stress.
            Defaults to 0.1 (according to OpenSeesPy documentation)
        """
        self.sectionsection = deepcopy(section)
        ranges = section.bars_ranges_position_y
        bars = section.bars_position_x
        wy = self.__Compute_w__Compute_w(ranges, section.D_bars)
        wx_top = self.__Compute_w__Compute_w(bars[0], section.D_bars)
        wx_bottom = self.__Compute_w__Compute_w(bars[-1], section.D_bars)

        super().__init__(ID, section.bc, section.dc, section.Ac, section.fc, section.Ec, section.nr_bars, section.D_bars,
            wx_top, wx_bottom, wy, section.s, section.D_hoops, section.rho_s_x, section.rho_s_y, section.fs,
            ec=ec, ecp=ecp, fct=fct, et=et, esu=esu, beta=beta)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()

    def __Compute_w(self, vector, D_bars):
        """
        Private method that converts information from the section passed to the format of the class ConfMander1988Rect.

```

```python
        @param vector (list): Vector with the information from the section.
        @param D_bars (float): Diameter of the bars.

        @returns list: Converted information.
        """
        l = len(vector)
        w = np.zeros(l-2)
        for i, elem in enumerate(vector[1:l-1]):
            w[i] = elem - D_bars
        return w


class ConfMander1988Circ(MaterialModels):
    """
    Class that stores funcions and material properties of a RC circular section
        with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.
    For more information about the empirical model for the computation of the parameters, see Mander et Al. 1988, Karthik and Mander 2011 and SIA 262:2012.

    @param MaterialModels: Parent abstract class.
    """
    def __init__(self, ID: int, bc, Ac, fc, Ec, nr_bars, D_bars, s, D_hoops, rho_s_vol, fs,
        ec = 1, ecp = 1, fct = -1, et = -1, esu = -1, beta = 0.1):
        """
        Constructor of the class.

        @param ID (int): Unique material model ID.
        @param bc (float): Width of the confined core (from the centerline of the hoops, according to Mander et Al. 1988).
        @param Ac (float): Area of the confined core (according to Mander et Al. 1988).
        @param fc (float): Compressive concrete yield strength (needs to be negative).
        @param Ec (float): Young modulus.
        @param nr_bars (float): Number of reinforcement (allow float for computing the equivalent nr_bars with different reinforcement areas).
        @param D_bars (float): Diameter of the vertical reinforcing bars.
        @param s (float): Vertical spacing between hoops.
        @param D_hoops (float): Diameter of hoops.
        @param rho_s_vol (float): Compute the ratio of the volume of transverse confining steel to the volume of confined concrete core.
        @param fs (float): Yield stress for the hoops.
        @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
        @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
        @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
        @param beta (float, optional): Loating point value defining the exponential curve parameter to define the residual stress.
            Defaults to 0.1 (according to OpenSeesPy documentation)

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: bc needs to be positive.
        @exception NegativeValue: Ac needs to be positive.
        @exception PositiveValue: fc needs to be negative.
        @exception NegativeValue: Ec needs to be positive.
        @exception NegativeValue: nr_bars needs to be positive.
        @exception NegativeValue: D_bars needs to be positive.
        @exception NegativeValue: s needs to be positive.
        @exception NegativeValue: D_hoops needs to be positive.
        @exception NegativeValue: rho_s_vol needs to be positive.
        @exception NegativeValue: fs needs to be positive.
        @exception PositiveValue: ec needs to be negative if different from 1.
        @exception PositiveValue: ecp needs to be negative if different from 1.
        @exception NegativeValue: fct needs to be positive if different from -1.
        @exception NegativeValue: et needs to be positive if different from -1.
        @exception NegativeValue: esu needs to be positive if different from -1.
        """
        # Check
        if ID < 0: raise NegativeValue()
        if bc < 0: raise NegativeValue()
        if Ac < 0: raise NegativeValue()
        if fc > 0: raise PositiveValue()
        if Ec < 0: raise NegativeValue()
        if nr_bars < 0: raise NegativeValue()
        if D_bars < 0: raise NegativeValue()
        if s < 0: raise NegativeValue()
        if D_hoops < 0: raise NegativeValue()
        if rho_s_vol < 0: raise NegativeValue()
        if fs < 0: raise NegativeValue()
        if ec != 1 and ec > 0: raise PositiveValue()
        if ecp != 1 and ecp > 0: raise PositiveValue()
        if fct != -1 and fct < 0: raise NegativeValue()
```

```
        if et != -1 and et < 0: raise NegativeValue()
        if esu != -1 and esu < 0: raise NegativeValue()

        # Arguments
        self.IDID = ID
        self.bcbc = bc
        self.AcAc = Ac
        self.fcfc = fc
        self.EcEc = Ec
        self.nr_barsnr_bars = nr_bars
        self.D_barsD_bars = D_bars
        self.ss = s
        self.D_hoopsD_hoops = D_hoops
        self.rho_s_volrho_s_vol = rho_s_vol
        self.fsfs = fs
        self.esuesu = 0.05 if esu == -1 else esu
        self.betabeta = beta

        # Initialized the parameters that are dependent from others
        self.section_name_tagsection_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit(ec, ecp, fct, et)

    def ReInit(self, ec = 1, ecp = 1, fct = -1, et = -1):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
        @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
        @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        """
        # Check applicability
        self.CheckApplicabilityCheckApplicabilityCheckApplicability()

        # Arguments
        self.ecec = self.Compute_ecCompute_ec() if ec == 1 else ec
        self.ecpecp = self.Compute_ecpCompute_ecp() if ecp == 1 else ecp
        self.fctfct = self.Compute_fctCompute_fct() if fct == -1 else fct
        self.etet = self.Compute_etCompute_et() if et == -1 else et

        # Members
        s_prime = self.ss - self.D_hoopsD_hoops
        self.ecuecu = self.Compute_ecuCompute_ecu()
        self.AeAe = math.pi/4 * (self.bcbc - s_prime/2)**2
        self.rho_ccrho_cc = self.nr_barsnr_bars*self.D_barsD_bars**2/4.0*math.pi / self.AcAc
        self.AccAcc = self.AcAc*(1.0-self.rho_ccrho_cc)
        self.keke = self.AeAe/self.AccAcc
        self.flfl = -self.rho_s_volrho_s_vol * self.fsfs / 2
        self.fl_primefl_prime = self.flfl * self.keke
        self.K_comboK_combo = -1.254 + 2.254 * math.sqrt(1.0+7.94*self.fl_primefl_prime/self.fcfc) - 2.0*self.fl_primefl_prime/self.fcfc
        self.fccfcc = self.fcfc * self.K_comboK_combo
        self.eccecc = self.Compute_eccCompute_ecc()
        self.eccueccu = self.Compute_eccuCompute_eccu()
        if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag = self.section_name_tagsection_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    # Methods
    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "ConfMander1988Circ"], # Tag for differentiating different data
            ["ID", self.IDID],
            ["section_name_tag", self.section_name_tagsection_name_tag],
            ["bc", self.bcbc],
            ["Ac", self.AcAc],
            ["fc", self.fcfc],
            ["Ec", self.EcEc],
            ["ec", self.ecec],
            ["ecp", self.ecpecp],
            ["ecu", self.ecuecu],
            ["fct", self.fctfct],
            ["et", self.etet],
            ["fcc", self.fccfcc],
            ["ecc", self.eccecc],
```

```
            ["eccu", self.eccueccu],
            ["beta", self.betabeta],
            ["nr_bars", self.nr_barsnr_bars],
            ["D_bars", self.D_barsD_bars],
            ["s", self.ss],
            ["D_hoops", self.D_hoopsD_hoops],
            ["rho_s_vol", self.rho_s_volrho_s_vol],
            ["fs", self.fsfs],
            ["esu", self.esuesu],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self, plot = False, block = False, concrete04 = True):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        @param concrete04 (bool, optional): Option to show in the plot the concrete04 or concrete01 if False. Defaults to True.
        """
        print("")
        print("Requested info for Confined Mander 1988 (circular) material model Parameters, ID = {}".format(self.IDID))
        print("Section associated: {} ".format(self.section_name_tagsection_name_tag))
        print('Concrete strength fc = {} MPa'.format(self.fcfc/MPa_unit))
        print('Concrete strength confined fcc = {} MPa'.format(self.fccfcc/MPa_unit))
        print('Strain at maximal strength ec = {}'.format(self.ecec))
        print('Strain at maximal strength confined ecc = {}'.format(self.eccecc))
        print('Maximal strain ecu = {}'.format(self.ecuecu))
        print('Maximal strain confined eccu = {}'.format(self.eccueccu))
        print("")

        if plot:
            fig, ax = plt.subplots()
            if concrete04:
                PlotConcrete04(self.fccfcc, self.EcEc, self.eccecc, self.eccueccu, "C", ax, self.IDID)
            else:
                PlotConcrete01(self.fccfcc, self.eccecc, 0.0, self.eccueccu, ax, self.IDID)

            if block:
                plt.show()


    def CheckApplicability(self):
        """
        Implementation of the homonym abstract method.
        See parent class MaterialModels for detailed information.
        """
        Check = True
        if self.fcfc < -110*MPa_unit: # Deierlein 1999
            Check = False
            print("With High Strength concrete (< -110 MPa), a better material model should be used (see Abdesselam et Al. 2019")
        if not Check:
            print("The validity of the equations is not fullfilled.")
            print("!!!!!!! WARNING !!!!!!! Check material model of Confined Mander 1988, ID=", self.IDID)
            print("")


    def Compute_ec(self):
        """
        Method that computes the compressive concrete yield strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        # return -0.002 # Alternative: Mander et Al. 1988
        return -0.0015 + self.fcfc/MPa_unit/70000 # Karthik Mander 2011


    def Compute_ecp(self):
        """
        Method that computes the compressive concrete spalling strain.
        For more information, see Mander et Al. 1988.

        @returns float: Strain
        """
        return 2.0*self.ecec


    def Compute_fct(self):
        """
```

```
        Method that computes the tensile concrete yield stress.
        For more information, see SIA 262:2012. Assume that the confinement do not play an essential role in tension.

        @returns float: Stress.
        """
        return 0.30 * math.pow(-self.fcfc/MPa_unit, 2/3) * MPa_unit


    def Compute_et(self):
        """
        Method that computes the tensile concrete yield strain.
        For more information, see Mander et Al. 1988 (eq 45).

        @returns float: Strain.
        """
        return self.fctfct/self.EcEc


    def Compute_ecu(self):
        """
        Method that computes the compressive concrete failure strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        # return -0.004 # Alternative: Mander et Al. 1988
        return -0.012 - 0.0001 * self.fcfc/MPa_unit # Karthik Mander 2011


    def Compute_ecc(self):
        """
        Method that computes the compressive confined concrete yield strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        return (1.0 + 5.0 * (self.K_comboK_combo-1.0)) * self.ecec # Karthik Mander 2011


    def Compute_eccu(self):
        """
        Method that computes the compressive confined concrete failure strain.
        For more information, see Karthik and Mander 2011.

        @returns float: Strain
        """
        # return -0.004 + (1.4*(self.rho_s_x+self.rho_s_y)*self.esu*self.fs) / self.fcc # Alternative: Prof. Katrin Beyer
        return 5*self.eccecc # Karthik Mander 2011


    def Concrete01(self):
        """
        Generate the material model Concrete01 for rectangular section confined concrete (Mander 1988).
        See _Concrete01 function for more information. Use this method or Concrete04, not both (only one material model for ID).
        """
        _Concrete01(self.IDID, self.eccecc, self.fccfcc, self.eccueccu)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


    def Concrete04(self):
        """
        Generate the material model Concrete04 for circular section confined concrete (Mander 1988).
        See _Concrete04 function for more information. Use this method or Concrete01, not both (only one material model for ID).
        """
        _Concrete04(self.IDID, self.fccfcc, self.eccecc, self.eccueccu, self.EcEc, self.fctfct, self.etet, self.betabeta)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


class ConfMander1988CircRCCircShape(ConfMander1988Circ):
    """
    Class that is the children of ConfMander1988Circ and combine the class RCCircShape (section) to retrieve the information needed.

    @param ConfMander1988Circ: Parent class.
    """
    def __init__(self, ID: int, section: RCCircShape, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1):
        """
        Constructor of the class. It passes the arguments into the parent class to generate the combination of the parent class
```

```
            and the section class RCCircShape.
        The copy of the section passed is stored in the member variable self.sectionsection.

        @param ID (int): Unique material model ID.
        @param section (RCCircShape): RCCircShape section object.
        @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
        @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
        @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
        @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
        @param beta (float, optional): Loating point value defining the exponential curve parameter to define the residual stress.
            Defaults to 0.1 (according to OpenSeesPy documentation)
        """
        self.sectionsection = deepcopy(section)
        super().__init__(ID, section.bc, section.Ac, section.fc, section.Ec, section.n_bars, section.D_bars, section.s, section.D_hoops,
            section.rho_s_vol, section.fs, ec=ec, ecp=ecp, fct=fct, et=et, esu=esu, beta=beta)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()


class UniaxialBilinear(MaterialModels):
    """
    Class that stores funcions and material properties of a simple uniaxial bilinear model
        with the OpenSeesPy command type used to model it is Steel01.

    @param MaterialModels: Parent abstract class.
    """
    def __init__(self, ID: int, fy, Ey, b = 0.01):
        """
        Constructor of the class.

        @param ID (int): Unique material model ID.
        @param fy (float): Yield stress.
        @param Ey (float): Young modulus.
        @param b (float, optional): Strain hardening factor. Defaults to 0.01.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: fy needs to be positive.
        @exception NegativeValue: Ey needs to be positive.
        """
        # Check
        if ID < 1: raise NegativeValue()
        if fy < 0: raise NegativeValue()
        if Ey < 0: raise NegativeValue()

        # Arguments
        self.IDID = ID
        self.fyfy = fy
        self.EyEy = Ey
        self.bb = b

        # Initialized the parameters that are dependent from others
        self.section_name_tagsection_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit()

    def ReInit(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        # Check applicability
        self.CheckApplicabilityCheckApplicabilityCheckApplicability()

        # Members
        self.eyey = self.fyfy / self.EyEy
        if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag = self.section_name_tagsection_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    # Methods
    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "UniaxialBilinear"], # Tag for differentiating different data
```

```
            ["ID", self.IDID],
            ["section_name_tag", self.section_name_tagsection_name_tag],
            ["fy", self.fyfy],
            ["Ey", self.EyEy],
            ["ey", self.eyey],
            ["b", self.bb],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self, plot = False, block = False):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        """
        print("")
        print("Requested info for Uniaxial Bilinear material model Parameters, ID = {}".format(self.IDID))
        print("Section associated: {} ".format(self.section_name_tagsection_name_tag))
        print('Yielding stress fy = {} MPa'.format(self.fyfy/MPa_unit))
        print('Young modulus Ey = {} MPa'.format(self.EyEy/MPa_unit))
        print('Maximal elastic strain epsilon y = {}'.format(self.eyey))
        print('Hardening factor b = {}'.format(self.bb))
        print("")

        if plot:
            # Data for plotting
            e_pl = 10.0 * self.eyey # to show that if continues with this slope
            sigma_pl = self.bb * self.EyEy * e_pl

            x_axis = np.array([0.0, self.eyey, (self.eyey+e_pl)])*100
            y_axis = np.array([0.0, self.fyfy, (self.fyfy+sigma_pl)])/MPa_unit

            fig, ax = plt.subplots()
            ax.plot(x_axis, y_axis, 'k-')

            ax.set(xlabel='Strain [%]', ylabel='Stress [MPa]',
                title='Uniaxial Bilinear model for material ID={}'.format(self.IDID))
            ax.grid()

            if block:
                plt.show()


    def CheckApplicability(self):
        """
        Implementation of the homonym abstract method.
        See parent class MaterialModels for detailed information.
        """
        Check = True
        # if len(self.wy) == 0 or len(self.wx_top) == 0 or len(self.wx_bottom) == 0:
        #     Check = False
        #     print("Hypothesis of one bar per corner not fullfilled.")
        if not Check:
            print("The validity of the equations is not fullfilled.")
            print("!!!!!!! WARNING !!!!!!! Check material model of Uniaxial Bilinear, ID=", self.IDID)
            print("")


    def Steel01(self):
        """
        Generate the material model Steel01 uniaxial bilinear material model.
        See _Steel01 function for more information.
        """
        _Steel01(self.IDID, self.fyfy, self.EyEy, self.bb)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


class UniaxialBilinearSteelIShape(UniaxialBilinear):
    """
    Class that is the children of UniaxialBilinear and combine the class SteelIShape (section) to retrieve the information needed.

    @param UniaxialBilinear: Parent class.
    """
    def __init__(self, ID: int, section: SteelIShape, b=0.01):
        """
        Constructor of the class. It passes the arguments into the parent class to generate the combination of the parent class
            and the section class SteelIShape.
        The copy of the section passed is stored in the member variable self.sectionsection.

```

```
        @param ID (int): Unique material model ID.
        @param section (SteelIShape): SteelIShape section object.
        @param b (float, optional): Strain hardening factor. Defaults to 0.01.
        """
        self.sectionsection = deepcopy(section)
        super().__init__(ID, section.Fy, section.E, b=b)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()


class GMP1970(MaterialModels):
    """
    Class that stores funcions and material properties of the vertical steel reinforcement bars
        with Giuffré, Menegotto and Pinto 1970 as the material model and the OpenSeesPy command type used to model it is Steel02.
    For more information about the empirical model for the computation of the parameters, see Giuffré, Menegotto and Pinto 1970 and Carreno et Al. 2020.

    @param MaterialModels: Parent abstract class.
    """
    def __init__(self, ID: int, fy, Ey, b = 0.02, R0 = 20, cR1 = 0.9, cR2 = 0.08, a1 = 0.039, a2 = 1.0, a3 = 0.029, a4 = 1.0):
        """
        Constructor of the class. The parameters are suggested as exposed in Carreno et Al. 2020 but also the one suggested by OpenSeesPy documentation are reliable
            (b = 0.015, R0 = 10, cR1 = 0.925, cR2 = 0.15).

        @param ID (int): Unique material model ID.
        @param fy (float): Steel yield strength.
        @param Ey (float): Young modulus.
        @param b (float, optional): Strain-hardening ratio. Defaults to 0.02, according to Carreno et Al. 2020.
        @param R0 (int, optional): First parameter to control the transition from elastic to plastic branches. Defaults to 20, according to Carreno et Al. 2020.
        @param cR1 (float, optional): Second parameter to control the transition from elastic to plastic branches. Defaults to 0.9, according to Carreno et Al. 2020.
        @param cR2 (float, optional): Third parameter to control the transition from elastic to plastic branches. Defaults to 0.08, according to Carreno et Al. 2020.
        @param a1 (float, optional): Isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain.
            Defaults to 0.039, according to Carreno et Al. 2020.
        @param a2 (float, optional): Coupled with a1. Defaults to 1.0, according to Carreno et Al. 2020.
        @param a3 (float, optional): Isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain.
            Defaults to 0.029, according to Carreno et Al. 2020.
        @param a4 (float, optional): Coupled with a3. Defaults to 1.0, according to Carreno et Al. 2020.

        @exception NegativeValue: ID needs to be a positive integer.
        """
        # Check
        if ID < 1: raise NegativeValue()

        # Arguments
        self.IDID = ID
        self.fyfy = fy
        self.EyEy = Ey
        self.bb = b
        self.R0R0 = R0
        self.cR1cR1 = cR1
        self.cR2cR2 = cR2
        self.a1a1 = a1
        self.a2a2 = a2
        self.a3a3 = a3
        self.a4a4 = a4

        # Initialized the parameters that are dependent from others
        self.section_name_tagsection_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit()

    def ReInit(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        # Check applicability
        self.CheckApplicabilityCheckApplicabilityCheckApplicability()

        # Members
        if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag = self.section_name_tagsection_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()

```

```

    # Methods
    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "GMP1970"], # Tag for differentiating different data
            ["ID", self.IDID],
            ["section_name_tag", self.section_name_tagsection_name_tag],
            ["fy", self.fyfy],
            ["Ey", self.EyEy],
            ["b", self.bb],
            ["R0", self.R0R0],
            ["cR1", self.cR1cR1],
            ["cR2", self.cR2cR2],
            ["a1", self.a1a1],
            ["a2", self.a2a2],
            ["a3", self.a3a3],
            ["a4", self.a4a4],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        print("")
        print("Requested info for GMP1970 (Giuffré-Menegotto-Pinto) material model Parameters, ID = {}".format(self.IDID))
        print("Section associated: {} ".format(self.section_name_tagsection_name_tag))
        print("Yield stress fy = {} MPa".format(self.fyfy/MPa_unit))
        print("Young modulus Ey = {} MPa".format(self.EyEy/MPa_unit))
        print("Strain hardening ratio b = {}".format(self.bb))
        print("Bauschinger effect factors R0 = {}, cR1 = {} and cR2 = {}".format(self.R0R0, self.cR1cR1, self.cR2cR2))
        print("Isotropic hardening factors a1 = {}, a2 = {}, a3 = {} and a4 = {}".format(self.a1a1, self.a2a2, self.a3a3, self.a4a4))
        print("")

        #TODO: add plot option (difficult to implement)


    def CheckApplicability(self):
        """
        Implementation of the homonym abstract method.
        See parent class MaterialModels for detailed information.
        """
        Check = True
        # No checks
        if not Check:
            print("The validity of the equations is not fullfilled.")
            print("!!!!!!! WARNING !!!!!!! Check material model of GMP1970, ID=", self.IDID)
            print("")


    def Steel02(self):
        """
        Generate the material model Steel02 uniaxial Giuffre-Menegotto-Pinto steel material with isotropic strain hardening.
        See _Steel02 function for more information.
        """
        _Steel02(self.IDID, self.fyfy, self.EyEy, self.bb, self.R0R0, self.cR1cR1, self.cR2cR2, self.a1a1, self.a2a2, self.a3a3, self.a4a4)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


class GMP1970RCRectShape(GMP1970):
    """
    Class that is the children of GMP1970 and combine the class RCRectShape (section) to retrieve the information needed.

    @param GMP1970: Parent class.
    """
    def __init__(self, ID: int, section: RCRectShape, b=0.02, R0=20.0, cR1=0.9, cR2=0.08, a1=0.039, a2=1.0, a3=0.029, a4=1.0):
        """
        Constructor of the class. It passes the arguments into the parent class to generate the combination of the parent class
            and the section class RCRectShape.
        The copy of the section passed is stored in the member variable self.sectionsection.

        @param ID (int): Unique material model ID.
        @param section (RCRectShape): RCRectShape section object.
        @param b (float, optional): Strain-hardening ratio. Defaults to 0.02, according to Carreno et
```

```
Al. 2020.
        @param R0 (int, optional): First parameter to control the transition from elastic to plastic branches. Defaults to 20, according to Carreno et Al. 2020.
        @param cR1 (float, optional): Second parameter to control the transition from elastic to plastic branches. Defaults to 0.9, according to Carreno et Al. 2020.
        @param cR2 (float, optional): Third parameter to control the transition from elastic to plastic branches. Defaults to 0.08, according to Carreno et Al. 2020.
        @param a1 (float, optional): Isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain.
            Defaults to 0.039, according to Carreno et Al. 2020.
        @param a2 (float, optional): Coupled with a1. Defaults to 1.0, according to Carreno et Al. 2020.
        @param a3 (float, optional): Isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain.
            Defaults to 0.029, according to Carreno et Al. 2020.
        @param a4 (float, optional): Coupled with a3. Defaults to 1.0, according to Carreno et Al. 2020.
        """
        self.sectionsection = deepcopy(section)
        super().__init__(ID, section.fy, section.Ey, b=b, R0=R0, cR1=cR1, cR2=cR2, a1=a1, a2=a2, a3=a3, a4=a4)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()


class UVC(MaterialModels):
    """
    Class that stores funcions and material properties of a steel profile or reinforcing bar
        with Updated Voce-Chaboche as the material model and the OpenSeesPy command type used to model it is UVCuniaxial.
    For more information about the how to calibrate the set of parameters, see
        de Castro e Sousa, Suzuki and Lignos 2020 and Hartloper, de Castro e Sousa and Lignos 2021.

    @param MaterialModels: Parent abstract class.
    """
    def __init__(self, ID: int, fy, Ey, QInf, b, DInf, a, cK: np.ndarray, gammaK: np.ndarray):
        """
        Constructor of the class.

        @param ID (int): Unique material model ID.
        @param fy (float): Initial yield stress of the steel material.
        @param Ey (float): Elastic modulus of the steel material.
        @param QInf (float): Maximum increase in yield stress due to cyclic hardening (isotropic hardening).
        @param b (float): Saturation rate of QInf.
        @param DInf (float): Decrease in the initial yield stress, to neglect the model updates set DInf = 0.
        @param a (float): Saturation rate of DInf, a > 0. If DInf == 0, then a is arbitrary (but still a > 0).
        @param cK (np.ndarray): Array of 1 dimension; each entry is one kinematic hardening parameter associated with one backstress, up to 8 may be specified.
        @param gammaK (np.ndarray): Array of 1 dimension; each entry is one saturation rate of kinematic hardening associated with one backstress, up to 8 may be specified.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: fy needs to be positive.
        @exception NegativeValue: Ey needs to be positive.
        @exception NegativeValue: QInf needs to be positive.
        @exception NegativeValue: b needs to be positive.
        @exception NegativeValue: DInf needs to be positive.
        @exception NegativeValue: a needs to be positive.
        @exception WrongArgument: cK can't be empty.
        @exception WrongArgument: cK and gammaK have as many entries as the number of backstresses (thus they have the same length).
        """
        # Check
        if ID < 1: raise NegativeValue()
        if fy < 0: raise NegativeValue()
        if Ey < 0: raise NegativeValue()
        if QInf < 0: raise NegativeValue()
        if b < 0: raise NegativeValue()
        if DInf < 0: raise NegativeValue()
        if a < 0: raise NegativeValue()
        if len(cK) == 0: raise WrongArgument()
        if len(cK) != len(gammaK): raise WrongArgument()
        if len(cK) != 2: print("!!!!!!! WARNING !!!!!!! Number of backstresses should be 2 for optimal performances")
        if DInf == 0: print("!!!!!!! WARNING !!!!!!! With DInf = 0, the model used is Voce-Chaboche (VC) not updated (UVC)")

        # Arguments
        self.IDID = ID
        self.fyfy = fy
        self.EyEy = Ey
        self.QInfQInf = QInf
        self.bb = b
        self.DInfDInf = DInf
```

```
        self.aa = a
        self.cKcK = copy(cK)
        self.gammaKgammaK = copy(gammaK)

        # Initialized the parameters that are dependent from others
        self.section_name_tagsection_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit()

    def ReInit(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        # Check applicability
        self.CheckApplicabilityCheckApplicabilityCheckApplicability()

        # Members
        self.NN = len(self.cKcK)
        if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag = self.section_name_tagsection_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    # Methods
    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "UVC"], # Tag for differentiating different data
            ["ID", self.IDID],
            ["section_name_tag", self.section_name_tagsection_name_tag],
            ["fy", self.fyfy],
            ["Ey", self.EyEy],
            ["QInf", self.QInfQInf],
            ["b", self.bb],
            ["DInf", self.DInfDInf],
            ["a", self.aa],
            ["N", self.NN],
            ["ck", self.cKcK],
            ["gammaK", self.gammaKgammaK],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        print("")
        print("Requested info for UVC material model Parameters, ID = {}".format(self.IDID))
        print("Section associated: {} ".format(self.section_name_tagsection_name_tag))
        print("Yield strength fy = {} MPa".format(self.fyfy/MPa_unit))
        print("Young modulus Ey = {} MPa".format(self.EyEy/MPa_unit))
        print("Isotropic hardening factor QInf = {} MPa and saturation rate  b = {}".format(self.QInfQInf/MPa_unit, self.bb))
        print("Decrease the initial yield stress DInf = {} MPa and saturation rate a = {}".format(self.DInfDInf/MPa_unit, self.aa))
        print("Kinematic hardening vector ({} backstresses) cK = {} MPa".format(self.NN, self.cKcK/MPa_unit))
        print("And associated saturation rate gammaK = {}".format(self.gammaKgammaK))
        print("")

        #TODO: implement plot (too complex for now)


    def CheckApplicability(self):
        """
        Implementation of the homonym abstract method.
        See parent class MaterialModels for detailed information.
        """
        Check = True
        # No checks
        if not Check:
            print("The validity of the equations is not fullfilled.")
            print("!!!!!!! WARNING !!!!!!! Check material model of UVC, ID=", self.IDID)
            print("")


    def UVCuniaxial(self):
        """
        Generate the material model Updated Voce-Chaboche (UVC) for uniaxial stress states.
        See _UVCuniaxial function for more information.
        """
```

```
        _UVCuniaxial(self.IDID, self.EyEy, self.fyfy, self.QInfQInf, self.bb, self.DInfDInf, self.aa, self.NN, self.cKcK, self.gammaKgammaK)
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


class UVCCalibrated(UVC):
    """
    Class that is the children of UVC that retrieve calibrated parameters from UVC_calibrated_parameters.txt.
        The file text can be modified by adding more calibrated parameters.

    @param UVC: Parent class.
    """
    def __init__(self, ID: int, calibration: str, fy = -1, E = -1):
        """
        Constructor of the class. It retrieve the parameters from UVC_calibrated_parameters.txt and pass them in the parent class.

        @param ID (int): Unique material model ID.
        @param calibration (str): Label of the calibration parameter set. The options are: \n
        # 'S355J2_25mm_plate' \n
        # 'S355J2_50mm_plate' \n
        # 'S355J2_HEB500_flange' \n
        # 'S355J2_HEB500_web' \n
        # 'S460NL_25mm_plate' \n
        # 'S690QL_25mm_plate' \n
        # 'A992Gr50_W14X82_web' \n
        # 'A992Gr50_W14X82_flange' \n
        # 'A500GrB_HSS305X16' \n
        # 'BCP325_22mm_plate' \n
        # 'BCR295_HSS350X22' \n
        # 'HYP400_27mm_plate' \n
        @param fy (float, optional): Yield strength. Defaults to -1, e.g. taken equal to the one given in the calibration parameter set.
        @param E (float, optional): Young modulus. Defaults to -1, e.g. taken equal to the one given in the calibration parameter set.

        @exception NegativeValue: fy needs to be positive if different from -1.
        @exception NegativeValue: E needs to be positive if different from -1.
        @exception NameError: calibration needs to be equal to the label of one of the set of calibrated parameters.
        """
        if fy != -1 and fy < 0: raise NegativeValue()
        if E != -1 and E < 0: raise NegativeValue()

        self.calibrationcalibration = calibration

        # Structure of the data to be stored
        names = ["Material", "Ey", "fy", "QInf", "b","DInf", "a", "C1", "gamma1", "C2", "gamma2"]
        # Get the data
        __location__ = os.path.realpath(os.path.join(os.getcwd(), os.path.dirname(__file__)))
        UVC_data = np.genfromtxt(os.path.join(__location__, 'UVC_calibrated_parameters.txt'), dtype=None, skip_header=1, names = names, encoding='ascii', delimiter='\t')
        # Define the index (with the location of the correct set of parameters)
        index = UVC_data["Material"] == calibration
        fy = UVC_data["fy"][index][0]*MPa_unit if fy == -1 else fy
        E = UVC_data["Ey"][index][0]*GPa_unit if E == -1 else E
        # Check
        if not index.any(): raise NameError("No calibrated parameters with that name. Note that there are no spaces in the label.")

        # Assign arguments value
        super().__init__(ID, fy, E, UVC_data["QInf"][index][0]*MPa_unit, UVC_data["b"][index][0],
            UVC_data["DInf"][index][0]*MPa_unit, UVC_data["a"][index][0],
            np.array([UVC_data["C1"][index][0], UVC_data["C2"][index][0]])*MPa_unit,
            np.array([UVC_data["gamma1"][index][0], UVC_data["gamma2"][index][0]]))


class UVCCalibratedRCRectShape(UVCCalibrated):
    """
    Class that is the children of UVCCalibrated and combines the class RCRectShape (section) to retrieve the information needed.

    @param UVCCalibrated: Parent class.
    """
    def __init__(self, ID: int, section: RCRectShape, calibration = 'S460NL_25mm_plate'):
        """
        Constructor of the class.

        @param ID (int): Unique material model ID.
        @param section (RCRectShape): RCRectShape section object.
        @param calibration (str): Label of the calibration parameter set. The options are listed in UVCCalibrated.
        Defaults to 'S460NL_25mm_plate'. Change it accordingly to the steel rebars material properties.
        """
```

```
        self.sectionsection = deepcopy(section)
        super().__init__(ID, calibration, section.fy, section.Ey)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()


class UVCCalibratedRCCircShape(UVCCalibrated):
    """
    Class that is the children of UVCCalibrated and combine the class RCCircShape (section) to retrieve the information needed.

    @param UVCCalibrated: Parent class.
    """
    def __init__(self, ID: int, section: RCCircShape, calibration = 'S460NL_25mm_plate'):
        """
        Constructor of the class.

        @param ID (int): Unique material model ID.
        @param section (RCCircShape): RCCircShape section object.
        @param calibration (str, optional): Label of the calibration parameter set. The options are listed in UVCCalibrated.
            Defaults to 'S460NL_25mm_plate'. Change it accordingly to the steel rebars material properties.
        """
        self.sectionsection = deepcopy(section)
        super().__init__(ID, calibration, section.fy, section.Ey)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()


class UVCCalibratedSteelIShapeFlange(UVCCalibrated):
    """
    Class that is the children of UVCCalibrated and combine the class SteelIShape (section) to retrieve the information needed
        for the material model of the flange (often used fo the entire section).

    @param UVCCalibrated: Parent class.
    """
    def __init__(self, ID: int, section: SteelIShape, calibration = 'S355J2_HEB500_flange'):
        """
        Constructor of the class.

        @param ID (int): Unique material model ID.
        @param section (SteelIShape): SteelIShape section object.
        @param calibration (str, optional): Label of the calibration parameter set. The options are listed in UVCCalibrated.
            Defaults to 'S355J2_HEB500_flange'. Change it accordingly to the steel rebars material properties.
        """
        self.sectionsection = deepcopy(section)
        super().__init__(ID, calibration, section.Fy, section.E)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()


class UVCCalibratedSteelIShapeWeb(UVCCalibrated):
    """
    Class that is the children of UVCCalibrated and combine the class SteelIShape (section) to retrieve the information needed
        for the material model of the web.

    @param UVCCalibrated: Parent class.
    """
    def __init__(self, ID: int, section: SteelIShape, calibration = 'S355J2_HEB500_web'):
        """
        Constructor of the class.

        @param ID (int): Unique material model ID.
        @param section (SteelIShape): SteelIShape section object.
        @param calibration (str, optional): Label of the calibration parameter set. The options are listed in UVCCalibrated.
            Defaults to 'S355J2_HEB500_web'. Change it accordingly to the steel rebars material properties.
        """
        self.sectionsection = deepcopy(section)
        super().__init__(ID, calibration, section.Fy_web, section.E)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()


# Public functions
def Concrete04Funct(fc, discretized_eps, ec, Ec):
    """
    Function with the equation of the curve of the confined and unconfined concrete (Popovics 1973).

    @param fc (float): Compressive concrete yield stress (negative).
    @param discretized_eps (float): Variable strain.
```

```
    @param ec (float): Compressive concrete yield strain (negative).
    @param Ec (float): Concrete Young modulus.

    @returns float: Stress in function of variable strain.
    """
    x = discretized_eps/ec
    r = Ec / (Ec - fc/ec)
    return fc*x*r / (r-1+x**r)


def PlotConcrete04(fc, Ec, ec, ecu, Type: str, ax, ID = 0):
    """
    Function that plots the confined/unconfined Concrete04 stress-strain curve.

    @param fc (float): Compressive concrete yield strength (needs to be negative).
    @param Ec (float): Young modulus.
    @param ec (float): Compressive concrete yield strain.
    @param ecu (float): Compressive concrete failure strain (negative).
    @param Type (str): Type of concrete (confined = 'C', unconfined = 'U')
    @param ax (matplotlib.axes._subplots.AxesSubplot): The figure's wrapper.
    @param ID (int, optional): ID of the material model. Defaults to 0 (= not defined).

    @exception NameError:

    Example: to create the plot, call this line to pass the correct ax:
        fig, ax = plt.subplots()
    """
    if Type == "C":
        name = "Confined (Co04)"
    elif Type == "U":
        name = "Unconfined (Co04)"
    else:
        raise NameError("Type should be C or U (ID={})".format(ID))

    # Data for plotting
    N = 1000
    x_axis = np.zeros(N)
    y_axis = np.zeros(N)
    for i in range(N):
        x_axis[i] = i/N*ecu
        y_axis[i] = Concrete04Funct(fc, x_axis[i], ec, Ec)


    ax.plot(x_axis*100.0, y_axis/MPa_unit, 'k-', label = name)
    ax.set(xlabel='Strain [%]', ylabel='Stress [MPa]',
                title='Mander 1988 (Concrete04) material model (ID={})'.format(ID))
    plt.legend()
    plt.grid()


def Concrete01Funct(fc, ec, fpcu, ecu, discretized_eps):
    """
    Function with the equation of the curve of the Concrete01 model.
    For more information, see Kent-Scott-Park concrete material object with
        degraded linear unloading/reloading stiffness according to the work of Karsan-Jirsa and no
 tensile strength.

    @param fc (float): Compressive concrete yield stress (negative).
    @param ec (float): Compressive concrete yield strain (negative).
    @param fpcu (float): Concrete crushing strength (negative).
    @param ecu (float): Concrete strain at crushing strength (negative).
    @param discretized_eps (float): Variable strain.

    @returns float: Stress in function of variable strain.
    """
    if discretized_eps > ec:
        eta = discretized_eps/ec;
        return fc*(2*eta-eta*eta);
    else:
        Ttangent = (fc-fpcu)/(ec-ecu)
        return fc + Ttangent*(discretized_eps-ec);


def PlotConcrete01(fc, ec, fpcu, ecu, ax, ID = 0):
    """
    Function that plots the Concrete01 stress-strain curve.

    @param fc (float): Compressive concrete yield stress (negative).
    @param ec (float): Compressive concrete yield strain (negative).
    @param fpcu (float): Concrete crushing strength (negative).
    @param ecu (float): Concrete strain at crushing strength (negative).
    @param ax (matplotlib.axes._subplots.AxesSubplot): The figure's wrapper.
    @param ID (int, optional): ID of the material model. Defaults to 0 (= not defined).

    Example: to create the plot, call this line to pass the correct ax:
        fig, ax = plt.subplots()
    """
```

```

    # Data for plotting
    N = 1000
    x_axis = np.zeros(N)
    y_axis = np.zeros(N)
    for i in range(N):
        x_axis[i] = i/N*ecu
        y_axis[i] = Concrete01Funct(fc, ec, fpcu, ecu, x_axis[i])


    ax.plot(x_axis*100.0, y_axis/MPa_unit, 'k--', label = "Co01")
    ax.set(xlabel='Strain [%]', ylabel='Stress [MPa]',
                    title='Mander 1988 (Concrete01) material model (ID={})'.format(ID))
    plt.legend()
    plt.grid()


# Private functions
def _Bilin(ID, Ke, a_s, My_star, theta_p, theta_pc, K, theta_u, rate_det):
    """
    Private function that generates the material model Bilin.
    OpenSeesPy command: \n
    uniaxialMaterial("Bilin", IDMat, K, asPos, asNeg, MyPos, MyNeg, LS, LK, LA, LD, cS, cK, cA, cD, th_pP, th_pN, th_pcP, th_pcN, ResP, ResN, th_uP, th_uN, DP, DN) \n
    Parameters (see OpenSeesPy documentation for more information): \n
    ID          Material Identification (integer)
    K           Initial stiffness after the modification for n (see Ibarra and Krawinkler, 2005)
    asPos       Strain hardening ratio after n modification (see Ibarra and Krawinkler, 2005)
    asNeg       Strain hardening ratio after n modification (see Ibarra and Krawinkler, 2005)
    MyPos       Positive yield moment (with sign)
    MyNeg       Negative yield moment (with sign)
    LS = 1000   Basic strength deterioration parameter (see Lignos and Krawinkler, 2009) (a very large # = no cyclic deterioration)
    LK = 1000   Unloading stiffness deterioration parameter (see Lignos and Krawinkler, 2009) (a very large # = no cyclic deterioration)
    LA = 1000   Accelerated reloading stiffness deterioration parameter (see Lignos and Krawinkler, 2009) (a very large # = no cyclic deterioration)
    LD = 1000   Post-capping strength deterioration parameter (see Lignos and Krawinkler, 2009) (a very large # = no cyclic deterioration)
    cS = 1      Exponent for basic strength deterioration (c = 1.0 for no deterioration)
    cK = 1      Exponent for unloading stiffness deterioration (c = 1.0 for no deterioration)
    cA = 1      Exponent for accelerated reloading stiffness deterioration (c = 1.0 for no deterioration)
    cD = 1      Exponent for post-capping strength deterioration (c = 1.0 for no deterioration)
    th_pP       Plastic rotation capacity for positive loading direction (exemple 0.025)
    th_pN       Plastic rotation capacity for negative loading direction (exemple 0.025)
    th_pcP      Post-capping rotation capacity for positive loading direction (exemple 0.3)
    th_pcN      Post-capping rotation capacity for negative loading direction (exemple 0.3)
    KP          Residual strength ratio for positive loading direction (exemple 0.4)
    KN          Residual strength ratio for negative loading direction (exemple 0.4)
    th_uP       Ultimate rotation capacity for positive loading direction (exemple 0.4)
    th_uN       Ultimate rotation capacity for negative loading direction (exemple 0.4)
    rateDetP    Rate of cyclic deterioration for positive loading direction (exemple 1.0)
    rateDetN    Rate of cyclic deterioration for negative loading direction (exemple 1.0)
    """
    uniaxialMaterial("Bilin", ID, Ke, a_s, a_s, My_star, -1.0*My_star, 1., 1., 1., 1., 1., 1., 1., 1., theta_p, theta_p, theta_pc, theta_pc,
        K, K, theta_u, theta_u, rate_det, rate_det)


def _Hysteretic(ID, M1, gamma1, M2, gamma2, M3, gamma3, pinchx, pinchy, dmg1, dmg2, beta):
    """
    Private function that generates the material model Hysteretic.
    OpenSeesPy command: \n
    uniaxialMaterial('Hysteretic', matTag, *p1, *p2, *p3=p2, *n1, *n2, *n3=n2, pinchX, pinchY, damage1, damage2, beta=0.0) \n
    Parameters (see OpenSeesPy documentation for more information): \n
    matTag       integer tag identifying material
    p1           stress and strain (or force & deformation) at first point of the envelope in the positive direction
    p2           stress and strain (or force & deformation) at second point of the envelope in the positive direction
    p3           stress and strain (or force & deformation) at third point of the envelope in the positive direction (optional)
    n1           stress and strain (or force & deformation) at first point of the envelope in the negative direction
    n2           stress and strain (or force & deformation) at second point of the envelope in the negative direction
    n3           stress and strain (or force & deformation) at third point of the envelope in the negative direction (optional)
    pinchX       pinching factor for strain (or deformation) during reloading
    pinchY       pinching factor for stress (or force) during reloading
    damage1      damage due to ductility: D1(mu-1)
    damage2      damage due to energy: D2(Eii/Eult)
    beta         power used to determine the degraded unloading stiffness based on ductility, mu-beta (optional, default=0.0)
    """
```

```
    uniaxialMaterial("Hysteretic", ID, M1, gamma1, M2, gamma2, M3, gamma3, -M1, -gamma1, -M2, -gamma2, -M3, -gamma3,
        pinchx, pinchy, dmg1, dmg2, beta)


def _Concrete04(ID, fc, ec, ecu, Ec, fct, et, beta):
    """
    Private function that generates the material model Concrete04 Popovics Concrete material model.
    OpenSeesPy command: \n
    uniaxialMaterial("Concrete04", matTag, fc, ec, ecu, Ec, <fct et> <beta>) \n
    Parameters (see OpenSeesPy documentation for more information): \n
    matTag     integer tag identifying material
    fc      floating point values defining concrete compressive strength at 28 days (compression is negative)*
    ec      floating point values defining concrete strain at maximum strength*
    ecu     floating point values defining concrete strain at crushing strength*
    Ec      floating point values defining initial stiffness**
    fct     floating point value defining the maximum tensile strength of concrete
    et      floating point value defining ultimate tensile strain of concrete
    beta    loating point value defining the exponential curve parameter to define the residual stress (as a factor of ft) at etu
    """
    uniaxialMaterial("Concrete04", ID, fc, ec, ecu, Ec, fct, et, beta)


def _Concrete01(ID, ec, fc, ecu, fpcu = 0.0):
    """
    Private function that generates the material model Concrete02 concrete material model.
    OpenSeesPy command: \n
    uniaxialMaterial(’Concrete01’, matTag, fpc, epsc0, fpcu, epsU) \n
    Parameters (see OpenSeesPy documentation for more information): \n
    matTag  integer tag identifying material
    fpc     concrete compressive strength at 28 days (compression is negative)*
    epsc0   concrete strain at maximum strength*
    fpcu    concrete crushing strength *
    epsU    concrete strain at crushing strength*
    """
    uniaxialMaterial(’Concrete01’, ID, fc, ec, fpcu, ecu)


def _Steel01(ID, fy, Ey, b):
    """
    Private function that generates the material model Steel01 uniaxial bilinear steel material
        with kinematic hardening and optional isotropic hardening described by a non-linear evolution equation.
    OpenSeesPy command: \n
    uniaxialMaterial(’Steel01’, matTag, Fy, E0, b, a1, a2, a3, a4) \n
    Parameters (see OpenSeesPy documentation for more information): \n
    matTag  integer tag identifying material
    Fy  yield strength
    E0  initial elastic tangent
    b   strain-hardening ratio (ratio between post-yield tangent and initial elastic tangent)
    a1  isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain of a2*(Fy/E0). (optional)
    a2  isotropic hardening parameter (see explanation under a1). (optional).
    a3  isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain of a4*(Fy/E0). (optional)
    a4  isotropic hardening parameter (see explanation under a3). (optional)
    """
    uniaxialMaterial("Steel01", ID, fy, Ey, b)


def _Steel02(ID, fy, Ey, b, R0, cR1, cR2, a1, a2, a3, a4):
    """
    Private function that generates the material model Steel02 uniaxial Giuffre-Menegotto-Pinto steel material with isotropic strain hardening.
    OpenSeesPy command: \n
    uniaxialMaterial(’Steel02’, matTag, Fy, E, b, R0, cR1, cR2, a1, a2, a3, a4, sigInit) \n
    Parameters (see OpenSeesPy documentation for more information): \n
    matTag      Integer tag identifying material
    Fy          Yield strength
    E0          Initial elastic tangent
    b           Strain-hardening ratio (ratio between post-yield tangent and initial elastic tangent)
    R0 CR1 CR2  Parameters to control the transition from elastic to plastic branches.
    a1          Isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain of a2*(Fy/E0). (optional)
    a2          Isotropic hardening parameter (see explanation under a1). (optional default = 1.0).
    a3          Isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain of a4*(Fy/E0). (optional default = 0.0)
    a4          Isotropic hardening parameter (see explanation under a3). (optional default = 1.0)
    sigInit     Initial Stress Value (optional, default: 0.0) the strain is calculated from epsP=sigInit/E
                    if (sigInit!= 0.0) { double epsInit = sigInit/E; eps = trialStrain+epsInit; } else eps = trialStrain;
    """
    uniaxialMaterial(’Steel02’, ID, fy, Ey, b, R0, cR1, cR2, a1, a2, a3, a4)

```

```

def _UVCuniaxial(ID, Ey, fy, QInf, b, DInf, a, N, cK, gammaK):
    """
    Private function that generates the material model Updated Voce-Chaboche (UVC) material for uniaxial stress states.
    This material is a refined version of the classic nonlinear isotropic/kinematic hardening material model based on the Voce
        isotropic hardening law and the Chaboche kinematic hardening law.
    The UVC model contains an updated isotropic hardening law, with parameter constraints, to simulate the permanent decrease
        in yield stress with initial plastic loading associated with the discontinuous yielding phenomenon in mild steels.
    OpenSeesPy command: \n
    uniaxialMaterial('UVCuniaxial', matTag, E, fy, QInf, b, DInf, a, N, C1, gamma1, <C2 gamma2 C3 gamma3 ... C8 gamma8>) \n
    Parameters (see OpenSeesPy documentation for more information): \n
    matTag  Integer tag identifying the material.
    E       Elastic modulus of the steel material.
    fy      Initial yield stress of the steel material.
    QInf    Maximum increase in yield stress due to cyclic hardening (isotropic hardening).
    b       Saturation rate of QInf, b > 0.
    DInf    Decrease in the initial yield stress, to neglect the model updates set DInf = 0.
    a       Saturation rate of DInf, a > 0. If DInf == 0, then a is arbitrary (but still a > 0).
    N       Number of backstresses to define, N >= 1.
    cK      Kinematic hardening parameter associated with backstress component k (vector).
    gammaK  Saturation rate of kinematic hardening associated with backstress component k (vector).
    """
    backstresses = []
    for ii in range(N):
        backstresses.append(cK[ii])
        backstresses.append(gammaK[ii])
    uniaxialMaterial('UVCuniaxial', ID, Ey, fy, QInf, b, DInf, a, N, *backstresses)

```

8.19 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py File Reference

Classes

- class ElasticElement

    Class that handles the storage and manipulation of a elastic element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

- class ElasticElementSteelIShape

    Class that is the children of ElasticElement and combine the class SteelIShape (section) to retrieve the information needed.

- class ForceBasedElement

    Class that handles the storage and manipulation of a force-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

- class ForceBasedElementFibersCircRCCircShape

    Class that is the children of ForceBasedElement and combine the class FibersCircRCCircShape (fiber section) to retrieve the information needed.

- class ForceBasedElementFibersIShapeSteelIShape

    Class that is the children of ForceBasedElement and combine the class FibersIShapeSteelIShape (fiber section) to retrieve the information needed.

- class ForceBasedElementFibersRectRCRectShape

    Class that is the children of ForceBasedElement and combine the class FibersRectRCRectShape (fiber section) to retrieve the information needed.

- class GIFBElement

    Class that handles the storage and manipulation of a Gradient-Inelastic Flexibility-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

- class GIFBElementFibersCircRCCircShape

    Class that is the children of GIFBElement and combine the class FibersCircRCCircShape (fiber section) to retrieve the information needed.

- class GIFBElementFibersRectRCRectShape

Class that is the children of GIFBElement and combine the class FibersRectRCRectShape (fiber section) to retrieve the information needed.

- class GIFBElementRCCircShape

  Class that is the children of GIFBElement and combine the class RCCircShape (section) to retrieve the information needed.
- class GIFBElementRCRectShape

  Class that is the children of GIFBElement and combine the class RCRectShape (section) to retrieve the information needed.
- class MemberModel

  Parent abstract class for the storage and manipulation of a member's information (mechanical and geometrical parameters, etc) and the initialisation in the model.
- class PanelZone

  Class that handles the storage and manipulation of a panel zone's information (mechanical and geometrical parameters, etc) and the initialisation in the model.
- class PanelZoneRCS

  WIP: Class that is the children of PanelZone and it's used for the panel zone in a RCS (RC column continous, Steel beam).
- class PanelZoneSteelIShape

  Class that is the children of PanelZone and combine the class SteelIShape (section) to retrieve the information needed.
- class PanelZoneSteelIShapeGupta1999

  Class that is the children of PanelZoneSteelIShape and automatically create the spring material model Gupta 1999 (ID = master_node_ID).
- class PanelZoneSteelIShapeSkiadopoulos2021

  Class that is the children of PanelZoneSteelIShape and automatically create the spring material model Skiadopoulos 2021 (ID = master_node_ID).
- class SpringBasedElement

  Class that handles the storage and manipulation of a spring-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.
- class SpringBasedElementModifiedIMKSteelIShape

  Class that is the children of SpringBasedElement and combine the class SteelIShape (section) to retrieve the information needed.
- class SpringBasedElementSteelIShape

  Class that is the children of SpringBasedElement and combine the class SteelIShape (section) to retrieve the information needed.

Namespaces

- namespace MemberModel

  Module for the member model.

Functions

- def DefinePanelZoneElements (MasterNodeID, E, RigidA, RigidI, TransfID)

  Function that defines the 8 panel zone elements.
- def DefinePanelZoneNodes (int MasterNodeID, MidPanelZoneWidth, MidPanelZoneHeight)

  Function that defines the remaining 10 nodes of a panel zone given the dimensions and the master node (top center one).

8.20 MemberModel.py

Go to the documentation of this file.

```
"""
Module for the member model.
Carmine Schipani, 2021
"""

from openseespy.opensees import *
import matplotlib.pyplot as plt
import numpy as np
import os
import math
from abc import abstractmethod
from copy import copy, deepcopy
from OpenSeesPyAssistant.Section import *
from OpenSeesPyAssistant.DataManagement import *
from OpenSeesPyAssistant.ErrorHandling import *
from OpenSeesPyAssistant.Units import *
from OpenSeesPyAssistant.Constants import *
from OpenSeesPyAssistant.Fibers import *
from OpenSeesPyAssistant.Connections import *
from OpenSeesPyAssistant.FunctionalFeatures import *

class MemberModel(DataManagement):
    """
    Parent abstract class for the storage and manipulation of a member's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

    @param DataManagement: Parent abstract class.
    """
    @abstractmethod
    def Record(self, ele_ID, name_txt: str, data_dir: str, force_rec = True, def_rec = True, time_rec = True):
        """
        Abstract method that records the forces, deformation and time of the member associated with the class.

        @param ele_ID (int): The ID of the element that will be recorded.
        @param name_txt (str): Name of the recorded data (no .txt).
        @param data_dir (str): Directory for the storage of data.
        @param force_rec (bool, optional): Option to record the forces (Fx, Fy, Mz). Defaults to True.
        @param def_rec (bool, optional): Option to record the deformation (theta) for ZeroLength element. Defaults to True.
        @param time_rec (bool, optional): Option to record time. Defaults to True.
        """
        if self.Initialized:
            if not os.path.exists(data_dir):
                print("Folder {} not found in this directory; creating one".format(data_dir))
                os.makedirs(data_dir)

            if time_rec:
                if force_rec:
                    recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-time", "-ele", ele_ID, "force")
                if def_rec:
                    recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-time", "-ele", ele_ID, "deformation")
            else:
                if force_rec:
                    recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-ele", ele_ID, "force")
                if def_rec:
                    recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-ele", ele_ID, "deformation")
        else:
                print("The element is not initialized (node and/or elements not created), ID = {}".format(ele_ID))

    @abstractmethod
    def RecordNodeDef(self, iNode_ID: int, jNode_ID: int, name_txt: str, data_dir: str, time_rec = True):
        """
        Abstract method that records the deformation and time of the member's nodes associated with the class.

        @param iNode_ID (int): ID of the node i.
        @param jNode_ID (int): ID of the node j.
        @param name_txt (str): Name of the recorded data (no .txt).
        @param data_dir (str): Directory for the storage of data.
        @param time_rec (bool, optional): Option to record time. Defaults to True.
        """
        if self.Initialized:
            if not os.path.exists(data_dir):
                print("Folder {} not found in this directory; creating one".format(data_dir))
```

```
                os.makedirs(data_dir)

            if time_rec:
                recorder("Node", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-time", "-node", iNode_ID, jNode_ID, "-dof", 1, 2, 3, "disp")
            else:
                recorder("Node", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-node", iNode_ID, jNode_ID, "-dof", 1, 2, 3, "disp")
        else:
                print("The element is not initialized (node and/or elements not created), iNode ID = {}, jNode ID = {}".format(iNode_ID, jNode_ID))


    def _CheckL(self):
        """
        Private abstract method to check if the length of the line member is the same (with 1 cm of tolerance) with the length defined in the section used.
        """
        iNode = np.array(nodeCoord(self.iNode_ID))
        jNode = np.array(nodeCoord(self.jNode_ID))
        L = np.linalg.norm(iNode-jNode)
        if abs(L-self.section.L) > 1*cm_unit:
            print("!!!!!!! WARNING !!!!!!! The length declared in the section name '{}' (L={} m) is different from thelength of the element associated (ID={}, L ={}m)".format(
                    self.section.name_tag, L/m_unit, self.element_ID, self.section.L/m_unit))


class PanelZone(MemberModel):
    """
    Class that handles the storage and manipulation of a panel zone's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

    @param MemberModel: Parent abstract class.
    """
    def __init__(self, master_node_ID: int, mid_panel_zone_width, mid_panel_zone_height, E, A_rigid, I_rigid, geo_transf_ID: int, mat_ID: int, pin_corners = True):
        """
        Constructor of the class.

        @param master_node_ID (int): ID of the master node (central top node that should be a grid node).
        @param mid_panel_zone_width (float): Mid panel zone width.
        @param mid_panel_zone_height (float): Mid panel zone height.
        @param E (float): Young modulus.
        @param A_rigid (float): A very rigid area.
        @param I_rigid (float): A very rigid moment of inertia.
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param mat_ID (int): ID of the material model for the panel zone spring.
        @param pin_corners (bool, optional): Option to pin the corners (xy03/xy04, xy06/xy07, xy09/xy10) or not. Used for RCS models. Defaults to True.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: mid_panel_zone_width needs to be positive.
        @exception NegativeValue: mid_panel_zone_height needs to be positive.
        @exception NegativeValue: E needs to be positive.
        @exception NegativeValue: A_rigid needs to be positive.
        @exception NegativeValue: I_rigid needs to be positive.
        @exception NegativeValue: geo_tranf_ID needs to be a positive integer.
        @exception NegativeValue: mat_ID needs to be a positive integer.
        """
        # Check
        if master_node_ID < 1: raise NegativeValue()
        # if master_node_ID > 99: raise WrongNodeIDConvention(master_node_ID)
        if mid_panel_zone_width < 0: raise NegativeValue()
        if mid_panel_zone_height < 0: raise NegativeValue()
        if E < 0: raise NegativeValue()
        if A_rigid < 0: raise NegativeValue()
        if I_rigid < 0: raise NegativeValue()
        if geo_transf_ID > 1: raise NegativeValue()
        if mat_ID < 0: raise NegativeValue()

        # Arguments
        self.master_node_IDmaster_node_ID = master_node_ID
        self.mid_panel_zone_widthmid_panel_zone_width = mid_panel_zone_width
        self.mid_panel_zone_heightmid_panel_zone_height = mid_panel_zone_height
        self.EE = E
        self.A_rigidA_rigid = A_rigid
        self.I_rigidI_rigid = I_rigid
        self.geo_transf_IDgeo_transf_ID = geo_transf_ID
        self.mat_IDmat_ID = mat_ID
        self.pin_cornerspin_corners = pin_corners

        # Initialized the parameters that are dependent from others
        self.col_section_name_tagcol_section_name_tag = "None"
        self.beam_section_name_tagbeam_section_name_tag = "None"
        self.InitializedInitialized = False
```

```
        self.ReInitReInit()


    def ReInit(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        # Arguments
        self.spring_IDspring_ID = -1

        # Members
        if self.col_section_name_tagcol_section_name_tag != "None": self.col_section_name_tagcol_section_name_tag = self.col_section_name_tagcol_section_name_tag + " (modified)"
        if self.beam_section_name_tagbeam_section_name_tag != "None": self.beam_section_name_tagbeam_section_name_tag = self.beam_section_name_tagbeam_section_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    # Methods
    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "PanelZone"], # Tag for differentiating different data
            ["master_node_ID", self.master_node_IDmaster_node_ID],
            ["col_section_name_tag", self.col_section_name_tagcol_section_name_tag],
            ["beam_section_name_tag", self.beam_section_name_tagbeam_section_name_tag],
            ["mat_ID", self.mat_IDmat_ID],
            ["spring_ID", self.spring_IDspring_ID],
            ["mid_panel_zone_width", self.mid_panel_zone_widthmid_panel_zone_width],
            ["mid_panel_zone_height", self.mid_panel_zone_heightmid_panel_zone_height],
            ["E", self.EE],
            ["A_rigid", self.A_rigidA_rigid],
            ["I_rigid", self.I_rigidI_rigid],
            ["tranf_ID", self.geo_transf_IDgeo_transf_ID],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self, plot = False, block = False):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        """
        print("")
        print("Requested info for Panel Zone member model, master node ID = {}".format(self.master_node_IDmaster_node_ID))
        print("Section associated, column: {} ".format(self.col_section_name_tagcol_section_name_tag))
        print("Section associated, beam: {} ".format(self.beam_section_name_tagbeam_section_name_tag))
        print("Material model of the panel zone ID = {}".format(self.mat_IDmat_ID))
        print("Spring ID = {} (if -1, not defined yet)".format(self.spring_IDspring_ID))
        print("Mid panel zone width = {} mm".format(self.mid_panel_zone_widthmid_panel_zone_width/mm_unit))
        print("Mid panel zone height = {} mm".format(self.mid_panel_zone_heightmid_panel_zone_height/mm_unit))
        print("Young modulus E = {} GPa".format(self.EE/GPa_unit))
        print("Area of the elements (rigid) = {} mm2".format(self.A_rigidA_rigid/mm2_unit))
        print("Moment of inetia of the elements (strong axis, rigid) = {} mm4".format(self.I_rigidI_rigid/mm4_unit))
        print("Geometric transformation = {}".format(self.geo_transf_IDgeo_transf_ID))
        print("")

        if plot:
            if self.InitializedInitialized:
                plot_member(self.element_arrayelement_array, "Panel zone, ID = {}".format(self.master_node_IDmaster_node_ID))
                if block:
                    plt.show()
            else:
                print("The panel zone is not initialized (node and elements not created) for master node ID = {}".format(self.master_node_IDmaster_node_ID))


    def CreateMember(self):
        """
        Method that initialises the member by calling the OpenSeesPy commands through various functions.
```

```
        """
        # Define nodes
        DefinePanelZoneNodes(self.master_node_IDmaster_node_ID, self.mid_panel_zone_widthmid_panel_zone_width, self.mid_panel_zone_heightmid_panel_zone_height)
        xy1 = IDConvention(self.master_node_IDmaster_node_ID, 1)
        xy01 = IDConvention(self.master_node_IDmaster_node_ID, 1, 1)
        xy03 = IDConvention(self.master_node_IDmaster_node_ID, 3, 1)
        xy04 = IDConvention(self.master_node_IDmaster_node_ID, 4, 1)
        xy06 = IDConvention(self.master_node_IDmaster_node_ID, 6, 1)
        xy07 = IDConvention(self.master_node_IDmaster_node_ID, 7, 1)
        xy09 = IDConvention(self.master_node_IDmaster_node_ID, 9, 1)
        xy10 = IDConvention(self.master_node_IDmaster_node_ID, 10)

        # Define rigid elements
        self.element_arrayelement_array = DefinePanelZoneElements(self.master_node_IDmaster_node_ID, self.EE, self.A_rigidA_rigid, self.I_rigidI_rigid, self.geo_transf_IDgeo_transf_ID)

        # Define zero length element
        self.spring_IDspring_ID = IDConvention(xy1, xy01)
        RotationalSpring(self.spring_IDspring_ID, xy1, xy01, self.mat_IDmat_ID)
        self.element_arrayelement_array.append([self.spring_IDspring_ID, xy1, xy01])
        self.iNode_IDiNode_ID = xy1
        self.jNode_IDjNode_ID = xy01

        # Pin connections
        if self.pin_cornerspin_corners:
            Pin(xy03, xy04)
            Pin(xy06, xy07)
            Pin(xy09, xy10)

        # Update class
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


    def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
        """
        Implementation of the homonym abstract method.
        See parent class MemberModel for detailed information.
        """
        super().Record(self.spring_IDspring_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec, time_rec=time_rec)


    def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
        """
        Implementation of the homonym abstract method.
        See parent class MemberModel for detailed information.
        """
        super().RecordNodeDef(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID, name_txt, data_dir, time_rec=time_rec)


    def _CheckL(self):
        """
        (placeholder). No applicable for the panel zone.
        """
        print("No length check for panel zone")


class PanelZoneSteelIShape(PanelZone):
    """
    Class that is the children of PanelZone and combine the class SteelIShape (section) to retrieve the information needed.

    @param PanelZone: Parent class.
    """
    def __init__(self, master_node_ID: int, col: SteelIShape, beam: SteelIShape, geo_transf_ID: int, mat_ID: int, rigid = RIGID):
        """
        Constructor of the class.

        @param master_node_ID (int): ID of the master node (central top node that should be a grid node).
        @param col (SteelIShape): SteelIShape column section object.
        @param beam (SteelIShape): SteelIShape beam section object.
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param mat_ID (int): ID of the material model for the panel zone spring.
        @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one element
            but enough small to avoid convergence problem. Defaults to RIGID.
        """
        self.colcol = deepcopy(col)
        self.beambeam = deepcopy(beam)
        super().__init__(master_node_ID, col.d/2.0, beam.d/2.0, col.E, max(col.A, beam.A)*rigid, max(col.Iy, beam.Iy)*rigid, geo_transf_ID, mat_ID)
```

```

        self.col_section_name_tagcol_section_name_tagcol_section_name_tag = col.name_tag
        self.beam_section_name_tagbeam_section_name_tagbeam_section_name_tag = beam.name_tag
        self.UpdateStoredDataUpdateStoredData()


class PanelZoneRCS(PanelZone):
    """
    WIP: Class that is the children of PanelZone and it’s used for the panel zone in a RCS (RC column continous, Steel beam).
    Note that the corners are not pinned (do it manually).

    @param PanelZone: Parent class.
    """
    def __init__(self, master_node_ID: int, col: RCRectShape, beam: SteelIShape, geo_transf_ID: int, mat_ID: int, rigid = RIGID):
        """
        Constructor of the class.

        @param master_node_ID (int): ID of the master node (central top node that should be a grid node).
        @param col (RCRectShape): RCRectShape column section object.
        @param beam (SteelIShape): SteelIShape beam section object.
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param mat_ID (int): ID of the material model for the panel zone spring.
        @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one element
            but enough small to avoid convergence problem. Defaults to RIGID.
        """
        self.colcol = deepcopy(col)
        self.beambeam = deepcopy(beam)
        super().__init__(master_node_ID, col.d/2.0, beam.d/2.0, beam.E, max(col.A, beam.A)*rigid, max(col.Iy, beam.Iy)*rigid, geo_transf_ID, mat_ID, False)

        self.col_section_name_tagcol_section_name_tagcol_section_name_tag = col.name_tag
        self.beam_section_name_tagbeam_section_name_tagbeam_section_name_tag = beam.name_tag
        self.UpdateStoredDataUpdateStoredData()


class PanelZoneSteelIShapeGupta1999(PanelZoneSteelIShape):
    """
    Class that is the children of PanelZoneSteelIShape and automatically create the spring material model Gupta 1999 (ID = master_node_ID).

    @param PanelZoneSteelIShape: Parent class.
    """
    def __init__(self, master_node_ID: int, col: SteelIShape, beam: SteelIShape, geo_transf_ID: int, t_dp = 0, rigid=RIGID):
        """
        Constructor of the class.

        @param master_node_ID (int): ID of the master node (central top node that should be a grid node).
        @param col (SteelIShape): SteelIShape column section object.
        @param beam (SteelIShape): SteelIShape beam section object.
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.
        @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one element
            but enough small to avoid convergence problem. Defaults to RIGID.
        """
        self.colcolcol = deepcopy(col)
        self.beambeambeam = deepcopy(beam)
        mat_ID = master_node_ID
        pz_spring = Gupta1999SteelIShape(mat_ID, col, beam, t_dp)
        pz_spring.Hysteretic()

        super().__init__(master_node_ID, col, beam, geo_transf_ID, mat_ID, rigid)


class PanelZoneSteelIShapeSkiadopoulos2021(PanelZoneSteelIShape):
    """
    Class that is the children of PanelZoneSteelIShape and automatically create the spring material model Skiadopoulos 2021 (ID = master_node_ID).

    @param PanelZoneSteelIShape: Parent class.
    """
    def __init__(self, master_node_ID: int, col: SteelIShape, beam: SteelIShape, geo_transf_ID: int, t_dp = 0, rigid=RIGID):
        """
        Constructor of the class.

        @param master_node_ID (int): ID of the master node (central top node that should be a grid node).
        @param col (SteelIShape): SteelIShape column section object.
```

```
        @param beam (SteelIShape): SteelIShape beam section object.
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.
        @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one element
            but enough small to avoid convergence problem. Defaults to RIGID.
        """
        self.colcolcol = deepcopy(col)
        self.beambeambeam = deepcopy(beam)
        mat_ID = master_node_ID
        pz_spring = Skiadopoulos2021SteelIShape(mat_ID, col, beam, t_dp)
        pz_spring.Hysteretic()

        super().__init__(master_node_ID, col, beam, geo_transf_ID, mat_ID, rigid)


def DefinePanelZoneNodes(MasterNodeID: int, MidPanelZoneWidth, MidPanelZoneHeight):
    """
    Function that defines the remaining 10 nodes of a panel zone given the dimensions and the master node (top center one).
    ID convention for the panel zone: \n
            PZNodeID:       12 nodes: top right 1xy (master), 1xy1 top right, 1xy09,1xy10     1xy      1xy1,1xy01 \n
                            clockwise 10 nodes xy01-xy10 (with double node at corners) o-----------o-----------o       \n
                            Spring at node 1xy1                                                        |           |       \n
            PZElemeneID:    8 elements: starting at node 1xy, clockwise                                |           |       \n
                            (see function DefinePanelZoneElements for more info)                       |           |       \n
                                                                                                       |           |       \n
                                                                                               1xy08 o           o 1xy02 \n
                                                                                                       |           |       \n
                                                                                                       |           |       \n
                                                                                                       |           |       \n
                                                                                                       |           |       \n
            o-----------o-----------o       \n
            1xy06,1xy07    1xy05    1xy03,1xy04 \n
        Note that the top right node is defined differently because is where the spring is.

    @param MasterNodeID (int): ID of the master node (central top node that should be a grid node).
    @param MidPanelZoneWidth (float): Mid panel zone width.
    @param MidPanelZoneHeight (float): Mid panel zone height.

    """
    # Get node coord and define useful variables
    m_node = np.array(nodeCoord(MasterNodeID))
    AxisCL = m_node[0]
    FloorCL = m_node[1] - MidPanelZoneHeight

    # Convention: Node of the spring (top right) is xy1
    node(IDConvention(MasterNodeID, 1), AxisCL+MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
    # Convention: Two notes in the corners (already defined one, xy1) clockwise from xy01 to xy10
    node(IDConvention(MasterNodeID, 1, 1), AxisCL+MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
    node(IDConvention(MasterNodeID, 2, 1), AxisCL+MidPanelZoneWidth, FloorCL)
    node(IDConvention(MasterNodeID, 3, 1), AxisCL+MidPanelZoneWidth, FloorCL-MidPanelZoneHeight)
    node(IDConvention(MasterNodeID, 4, 1), AxisCL+MidPanelZoneWidth, FloorCL-MidPanelZoneHeight)
    node(IDConvention(MasterNodeID, 5, 1), AxisCL, FloorCL-MidPanelZoneHeight)
    node(IDConvention(MasterNodeID, 6, 1), AxisCL-MidPanelZoneWidth, FloorCL-MidPanelZoneHeight)
    node(IDConvention(MasterNodeID, 7, 1), AxisCL-MidPanelZoneWidth, FloorCL-MidPanelZoneHeight)
    node(IDConvention(MasterNodeID, 8, 1), AxisCL-MidPanelZoneWidth, FloorCL)
    node(IDConvention(MasterNodeID, 9, 1), AxisCL-MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
    node(IDConvention(MasterNodeID, 10), AxisCL-MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)


def DefinePanelZoneElements(MasterNodeID, E, RigidA, RigidI, TransfID):
    """
    Function that defines the 8 panel zone elements. For the ID convention, see DefinePanelZoneNodes.

    @param MasterNodeID (int): ID of the master node (central top node that should be a grid node).
    @param E (float): Young modulus.
    @param RigidA (float): A very rigid area.
    @param RigidI (float): A very rigid moment of inertia.
    @param TransfID (int): The geometric transformation (for more information, see OpenSeesPy documentation).

    @returns list: List of lists, wth each list containing the ID of the element, of node i and node
```

```
j.
    """
    # Compute the ID of the nodes obeying to the convention used
    xy = MasterNodeID
    xy1 = IDConvention(xy, 1)
    xy01 = IDConvention(xy, 1, 1)
    xy02 = IDConvention(xy, 2, 1)
    xy03 = IDConvention(xy, 3, 1)
    xy04 = IDConvention(xy, 4, 1)
    xy05 = IDConvention(xy, 5, 1)
    xy06 = IDConvention(xy, 6, 1)
    xy07 = IDConvention(xy, 7, 1)
    xy08 = IDConvention(xy, 8, 1)
    xy09 = IDConvention(xy, 9, 1)
    xy10 = IDConvention(xy, 10)

    # Create element IDs using the convention: xy(a)xy(a)   with xy(a) = NodeID i and j
    #   Starting at MasterNodeID, clockwise
    # if MasterNodeID > 99:
    #     print("Warning, convention: MasterNodeID's digits should be 2")

    ele1 = IDConvention(xy, xy1)
    ele2 = IDConvention(xy01, xy02)
    ele3 = IDConvention(xy02, xy03)
    ele4 = IDConvention(xy04, xy05)
    ele5 = IDConvention(xy05, xy06)
    ele6 = IDConvention(xy07, xy08)
    ele7 = IDConvention(xy08, xy09)
    ele8 = IDConvention(xy10, xy)

    # Create panel zone elements
    #                             ID   ndI   ndJ      A    E     I     Transf
    element("elasticBeamColumn", ele1, xy,   xy1,  RigidA, E, RigidI, TransfID)
    element("elasticBeamColumn", ele2, xy01, xy02, RigidA, E, RigidI, TransfID)
    element("elasticBeamColumn", ele3, xy02, xy03, RigidA, E, RigidI, TransfID)
    element("elasticBeamColumn", ele4, xy04, xy05, RigidA, E, RigidI, TransfID)
    element("elasticBeamColumn", ele5, xy05, xy06, RigidA, E, RigidI, TransfID)
    element("elasticBeamColumn", ele6, xy07, xy08, RigidA, E, RigidI, TransfID)
    element("elasticBeamColumn", ele7, xy08, xy09, RigidA, E, RigidI, TransfID)
    element("elasticBeamColumn", ele8, xy10, xy, RigidA, E, RigidI, TransfID)

    # Create element array for forther manipulations
    element_array = [[ele1, xy, xy1],
        [ele2, xy01, xy02],
        [ele3, xy02, xy03],
        [ele4, xy04, xy05],
        [ele5, xy05, xy06],
        [ele6, xy07, xy08],
        [ele7, xy08, xy09],
        [ele8, xy10, xy]]

    return element_array


class ElasticElement(MemberModel):
    """
    Class that handles the storage and manipulation of a elastic element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

    @param MemberModel: Parent abstract class.
    """
    def __init__(self, iNode_ID: int, jNode_ID: int, A, E, Iy, geo_transf_ID: int, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param A (float): Area of the member.
        @param E (float): Young modulus.
        @param Iy (float): Second moment of inertia (strong axis).
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: A needs to be positive.
        @exception NegativeValue: E needs to be positive.
        @exception NegativeValue: Iy needs to be positive.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer.
        """
        # Check
        if iNode_ID < 1: raise NegativeValue()
        if jNode_ID < 1: raise NegativeValue()
        if A < 0: raise NegativeValue()
```

```
        if E < 0: raise NegativeValue()
        if Iy < 0: raise NegativeValue()
        if geo_transf_ID < 1: raise NegativeValue()
        if ele_ID != -1 and ele_ID < 1: raise NegativeValue()

        # Arguments
        self.iNode_IDiNode_ID = iNode_ID
        self.jNode_IDjNode_ID = jNode_ID
        self.AA = A
        self.EE = E
        self.IyIy = Iy
        self.geo_transf_IDgeo_transf_ID = geo_transf_ID

        # Initialized the parameters that are dependent from others
        self.section_name_tagsection_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit(ele_ID = -1)

    def ReInit(self, ele_ID = -1):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
        """
        # Members
        if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag = self.section_name_tagsection_name_tag + " (modified)"

        # element ID
        self.element_IDelement_ID = IDConvention(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID) if ele_ID == -1 else ele_ID

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    # Methods
    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "ElasticElement"], # Tag for differentiating different data
            ["element_ID", self.element_IDelement_ID],
            ["section_name_tag", self.section_name_tagsection_name_tag],
            ["A", self.AA],
            ["E", self.EE],
            ["Iy", self.IyIy],
            ["iNode_ID", self.iNode_IDiNode_ID],
            ["jNode_ID", self.jNode_IDjNode_ID],
            ["tranf_ID", self.geo_transf_IDgeo_transf_ID],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self, plot = False, block = False):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        """
        print("")
        print("Requested info for ElasticElement member model, ID = {}".format(self.element_IDelement_ID))
        print("Section associated {} ".format(self.section_name_tagsection_name_tag))
        print("Area A = {} mm2".format(self.AA/mm2_unit))
        print("Young modulus E = {} GPa".format(self.EE/GPa_unit))
        print("Moment of inertia Iy = {} mm4".format(self.IyIy/mm4_unit))
        print("Geometric transformation = {}".format(self.geo_transf_IDgeo_transf_ID))
        print("")

        if plot:
            if self.InitializedInitialized:
                plot_member(self.element_arrayelement_array, "Elastic Element, ID = {}".format(self.element_IDelement_ID))
                if block:
                    plt.show()
            else:
                print("The ElasticElement is not initialized (node and elements not created), ID = {}".format(self.element_IDelement_ID))


```

```
    def CreateMember(self):
        """
        Method that initialises the member by calling the OpenSeesPy commands through various functions.
        """
        self.element_arrayelement_array = [[self.element_IDelement_ID, self.iNode_IDiNode_ID, self.jNode_IDjNode_ID]]

        # Define element
        element("elasticBeamColumn", self.element_IDelement_ID, self.iNode_IDiNode_ID, self.jNode_IDjNode_ID, self.AA, self.EE, self.IyIy, self.geo_transf_IDgeo_transf_ID)

        # Update class
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


    def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
        """
        Implementation of the homonym abstract method.
        See parent class MemberModel for detailed information.
        """
        super().Record(self.element_IDelement_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec, time_rec=time_rec)


    def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
        """
        Implementation of the homonym abstract method.
        See parent class MemberModel for detailed information.
        """
        super().RecordNodeDef(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID, name_txt, data_dir, time_rec=time_rec)


class ElasticElementSteelIShape(ElasticElement):
    """
    Class that is the children of ElasticElement and combine the class SteelIShape (section) to retrieve the information needed.

    @param ElasticElement: Parent class.
    """
    def __init__(self, iNode_ID: int, jNode_ID: int, section: SteelIShape, geo_transf_ID: int, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param section (SteelIShape): SteelIShape section object.
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
        """
        self.sectionsection = deepcopy(section)
        super().__init__(iNode_ID, jNode_ID, section.A, section.E, section.Iy, geo_transf_ID, ele_ID)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()
        # Check length
        self._CheckL_CheckL()


class SpringBasedElement(MemberModel):
    """
    Class that handles the storage and manipulation of a spring-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

    @param MemberModel: Parent abstract class.
    """
    def __init__(self, iNode_ID: int, jNode_ID: int, A, E, Iy_mod, geo_transf_ID: int, mat_ID_i = -1, mat_ID_j = -1, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param A (float): Area of the member.
        @param E (float): Young modulus.
        @param Iy_mod (float): Second moment of inertia (strong axis).
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param mat_ID_i (int, optional): ID of the material model for the spring in the node i (if present). Defaults to -1.
        @param mat_ID_j (int, optional): ID of the material model for the spring in the node j (if present). Defaults to -1.
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
```

```
IDConvention to define it.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: A needs to be positive.
        @exception NegativeValue: E needs to be positive.
        @exception NegativeValue: Iy_mod needs to be positive.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer, if different from -1.
        @exception NegativeValue: ID needs to be a positive integer, if different from -1.
        @exception NameError: at least one spring needs to be defined.
        @exception NegativeValue: ID needs to be a positive integer, if different from -1.
        """
        # Check
        if iNode_ID < 1: raise NegativeValue()
        if jNode_ID < 1: raise NegativeValue()
        if A < 0: raise NegativeValue()
        if E < 0: raise NegativeValue()
        if Iy_mod < 0: raise NegativeValue()
        if geo_transf_ID < 1: raise NegativeValue()
        if mat_ID_i != -1 and mat_ID_i < 0: raise NegativeValue()
        if mat_ID_j != -1 and mat_ID_j < 0: raise NegativeValue()
        if mat_ID_i == -1 and mat_ID_j == -1: raise NameError("No springs defined for element ID =
{}".format(IDConvention(iNode_ID, jNode_ID)))
        if ele_ID != -1 and ele_ID < 0: raise NegativeValue()

        # Arguments
        self.iNode_IDiNode_ID = iNode_ID
        self.jNode_IDjNode_ID = jNode_ID
        self.AA = A
        self.EE = E
        self.Iy_modIy_mod = Iy_mod
        self.geo_transf_IDgeo_transf_ID = geo_transf_ID
        self.mat_ID_imat_ID_i = mat_ID_i
        self.mat_ID_jmat_ID_j = mat_ID_j

        # Initialized the parameters that are dependent from others
        self.section_name_tagsection_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit(ele_ID)


    def ReInit(self, ele_ID = -1):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
        """
        # Members
        if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
self.section_name_tagsection_name_tag + " (modified)"
        # orientation:
        self.ele_orientationele_orientation = NodesOrientation(self.iNode_IDiNode_ID,
self.jNode_IDjNode_ID)
        if self.ele_orientationele_orientation == "zero_length": raise
ZeroLength(IDConvention(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID))

        if self.mat_ID_imat_ID_i != -1:
            self.iNode_ID_springiNode_ID_spring = OffsetNodeIDConvention(self.iNode_IDiNode_ID,
self.ele_orientationele_orientation, "i")
        else:
            self.iNode_ID_springiNode_ID_spring = self.iNode_IDiNode_ID

        if self.mat_ID_jmat_ID_j != -1:
            self.jNode_ID_springjNode_ID_spring = OffsetNodeIDConvention(self.jNode_IDjNode_ID,
self.ele_orientationele_orientation, "j")
        else:
            self.jNode_ID_springjNode_ID_spring = self.jNode_IDjNode_ID

        # element ID
        self.element_IDelement_ID = IDConvention(self.iNode_ID_springiNode_ID_spring,
self.jNode_ID_springjNode_ID_spring) if ele_ID == -1 else ele_ID

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    # Methods
    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "SpringBasedElement"], # Tag for differentiating different data
            ["element_ID", self.element_IDelement_ID],
```

```
            ["section_name_tag", self.section_name_tagsection_name_tag],
            ["A", self.AA],
            ["E", self.EE],
            ["Iy_mod", self.Iy_modIy_mod],
            ["iNode_ID", self.iNode_IDiNode_ID],
            ["iNode_ID_spring", self.iNode_ID_springiNode_ID_spring],
            ["mat_ID_i", self.mat_ID_imat_ID_i],
            ["jNode_ID", self.jNode_IDjNode_ID],
            ["jNode_ID_spring", self.jNode_ID_springjNode_ID_spring],
            ["mat_ID_j", self.mat_ID_jmat_ID_j],
            ["ele_orientation", self.ele_orientationele_orientation],
            ["tranf_ID", self.geo_transf_IDgeo_transf_ID],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self, plot = False, block = False):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        """
        print("")
        print("Requested info for SpringBasedElement member model, ID = {}".format(self.element_IDelement_ID))
        print("Section associated {} ".format(self.section_name_tagsection_name_tag))
        print("Material model of the spring i, ID = {}".format(self.mat_ID_imat_ID_i))
        print("Material model of the spring j, ID = {}".format(self.mat_ID_jmat_ID_j))
        print("Area A = {} mm2".format(self.AA/mm2_unit))
        print("Young modulus E = {} GPa".format(self.EE/GPa_unit))
        print("n modified moment of inertia Iy_mod = {} mm4".format(self.Iy_modIy_mod/mm4_unit))
        print("Geometric transformation = {}".format(self.geo_transf_IDgeo_transf_ID))
        print("")

        if plot:
            if self.InitializedInitialized:
                plot_member(self.element_arrayelement_array, "SpringBased Element, ID = {}".format(self.element_IDelement_ID))
                if block:
                    plt.show()
            else:
                print("The SpringBasedElement is not initialized (node and elements not created), ID = {}".format(self.element_IDelement_ID))


    def CreateMember(self):
        """
        Method that initialises the member by calling the OpenSeesPy commands through various functions.
        """
        self.element_arrayelement_array = [[self.element_IDelement_ID, self.iNode_ID_springiNode_ID_spring, self.jNode_ID_springjNode_ID_spring]]
        if self.mat_ID_imat_ID_i != -1:
            # Define zero length element i
            node(self.iNode_ID_springiNode_ID_spring, *nodeCoord(self.iNode_IDiNode_ID))
            self.iSpring_IDiSpring_ID = IDConvention(self.iNode_IDiNode_ID, self.iNode_ID_springiNode_ID_spring)
            RotationalSpring(self.iSpring_IDiSpring_ID, self.iNode_IDiNode_ID, self.iNode_ID_springiNode_ID_spring, self.mat_ID_imat_ID_i)
            self.element_arrayelement_array.append([self.iSpring_IDiSpring_ID, self.iNode_IDiNode_ID, self.iNode_ID_springiNode_ID_spring])
        if self.mat_ID_jmat_ID_j != -1:
            # Define zero length element j
            node(self.jNode_ID_springjNode_ID_spring, *nodeCoord(self.jNode_IDjNode_ID))
            self.jSpring_IDjSpring_ID = IDConvention(self.jNode_IDjNode_ID, self.jNode_ID_springjNode_ID_spring)
            RotationalSpring(self.jSpring_IDjSpring_ID, self.jNode_IDjNode_ID, self.jNode_ID_springjNode_ID_spring, self.mat_ID_jmat_ID_j)
            self.element_arrayelement_array.append([self.jSpring_IDjSpring_ID, self.jNode_IDjNode_ID, self.jNode_ID_springjNode_ID_spring])

        # Define element
        element("elasticBeamColumn", self.element_IDelement_ID, self.iNode_ID_springiNode_ID_spring, self.jNode_ID_springjNode_ID_spring, self.AA, self.EE, self.Iy_modIy_mod, self.geo_transf_IDgeo_transf_ID)

        # Update class
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


    def Record(self, spring_or_element: str, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
        """
```

```
        Implementation of the homonym abstract method.
        See parent class MemberModel for detailed information.
        """
        if spring_or_element == "element":
            super().Record(self.element_IDelement_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec, time_rec=time_rec)
        elif spring_or_element == "spring_i":
            if self.mat_ID_imat_ID_i == -1:
                print("Spring i recorded not present in element ID = {}".format(self.element_IDelement_ID))
            else:
                super().Record(self.iSpring_IDiSpring_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec, time_rec=time_rec)
        elif spring_or_element == "spring_j":
            if self.mat_ID_jmat_ID_j == -1:
                print("Spring j recorded not present in element ID = {}".format(self.element_IDelement_ID))
            else:
                super().Record(self.jSpring_IDjSpring_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec, time_rec=time_rec)
        else:
            print("No recording option with: '{}' with element ID: {}".format(spring_or_element, self.element_IDelement_ID))


    def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
        """
        Implementation of the homonym abstract method.
        See parent class MemberModel for detailed information.
        """
        super().RecordNodeDef(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID, name_txt, data_dir, time_rec=time_rec)


class SpringBasedElementSteelIShape(SpringBasedElement):
    """
    Class that is the children of SpringBasedElement and combine the class SteelIShape (section) to retrieve the information needed.
    L_b is assumed the same for top and bottom springs.

    @param SpringBasedElement: Parent class.
    """
    def __init__(self, iNode_ID: int, jNode_ID: int, section: SteelIShape, geo_transf_ID: int, mat_ID_i=-1, mat_ID_j=-1, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param section (SteelIShape): SteelIShape section object.
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param mat_ID_i (int, optional): ID of the material model for the spring in the node i (if present). Defaults to -1.
        @param mat_ID_j (int, optional): ID of the material model for the spring in the node j (if present). Defaults to -1.
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NameError: at least one spring needs to be defined.
        @exception NegativeValue: ID needs to be a positive integer.
        """
        self.sectionsection = deepcopy(section)
        if mat_ID_i != -1 and mat_ID_i < 0: raise NegativeValue()
        if mat_ID_j != -1 and mat_ID_j < 0: raise NegativeValue()
        if mat_ID_i == -1 and mat_ID_j == -1: raise NameError("No springs defined for element ID = {}".format(IDConvention(iNode_ID, jNode_ID)))
        if ele_ID != -1 and ele_ID < 0: raise NegativeValue()

        super().__init__(iNode_ID, jNode_ID, section.A, section.E, section.Iy_mod, geo_transf_ID, mat_ID_i=mat_ID_i, mat_ID_j=mat_ID_j, ele_ID=ele_ID)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()
        # Check length
        self._CheckL_CheckL()


class SpringBasedElementModifiedIMKSteelIShape(SpringBasedElement):
    """
    Class that is the children of SpringBasedElement and combine the class SteelIShape (section) to retrieve the information needed.
    If there are two springs and the inflection point not in the middle, use two spring elements, connect them rigidly in the inflection point with one spring each in the extremes.
    L_b is assumed the same for top and bottom springs.

```

```
    @param SpringBasedElement: Parent class.
    """
    def __init__(self, iNode_ID: int, jNode_ID: int, section: SteelIShape, geo_transf_ID: int, new_mat_ID_i=-1, new_mat_ID_j=-1,
        N_G = 0, L_0 = -1, L_b = -1, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param section (SteelIShape): SteelIShape section object.
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param new_mat_ID_i (int, optional): New ID for the definition of the material model for the spring in the node i.
            If -1 is passed, the class generate no material model and no spring. If 0 is passed, no i spring. Defaults to -1.
        @param new_mat_ID_j (int, optional): New ID for the definition of the material model for the spring in the node j.
            If -1 is passed, the class generate no material model and no spring. If 0 is passed, no j spring. Defaults to -1.
        @param N_G (float, optional): Axial load. Defaults to 0.
        @param L_0 (float, optional): Distance from the maximal moment to zero. Defaults to -1, e.g. computed in __init__().
        @param L_b (float, optional): Maximal unbraced lateral buckling length. Defaults to -1, e.g. computed in __init__().
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NameError: at least one spring needs to be defined.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception ZeroLength: The two nodes are superimposed.
        """
        self.sectionsection = deepcopy(section)
        if new_mat_ID_i != -1 and new_mat_ID_i < 0: raise NegativeValue()
        if new_mat_ID_j != -1 and new_mat_ID_j < 0: raise NegativeValue()
        if new_mat_ID_i == 0 and new_mat_ID_j == 0: raise NameError("No springs imposed for element ID = {}. Use ElasticElement instead".format(IDConvention(iNode_ID, jNode_ID)))
        if ele_ID != -1 and ele_ID < 0: raise NegativeValue()

        if L_0 == -1:
            if new_mat_ID_i != 0 and new_mat_ID_j != 0:
                L_0 = section.L/2
            else:
                L_0 = section.L
        L_b = L_0 if L_b == -1 else L_b

        # auto assign ID for material of springs
        ele_orientation = NodesOrientation(iNode_ID, jNode_ID)
        if ele_orientation == "zero_length": raise ZeroLength(IDConvention(iNode_ID, jNode_ID))
        if new_mat_ID_i != 0 and new_mat_ID_i == -1:
            new_mat_ID_i = OffsetNodeIDConvention(iNode_ID, ele_orientation, "i")
        if new_mat_ID_j != 0 and new_mat_ID_j == -1:
            new_mat_ID_j = OffsetNodeIDConvention(jNode_ID, ele_orientation, "j")

        if new_mat_ID_i != 0:
            # Create mat i
            iSpring = ModifiedIMKSteelIShape(new_mat_ID_i, section, N_G, L_0 = L_0, L_b = L_b)
            iSpring.Bilin()

        if new_mat_ID_j != 0:
            # Create mat j
            jSpring = ModifiedIMKSteelIShape(new_mat_ID_j, section, N_G, L_0 = L_0, L_b = L_b)
            jSpring.Bilin()

        new_mat_ID_i = -1 if new_mat_ID_i == 0 else new_mat_ID_i
        new_mat_ID_j = -1 if new_mat_ID_j == 0 else new_mat_ID_j

        super().__init__(iNode_ID, jNode_ID, section.A, section.E, section.Iy_mod, geo_transf_ID, mat_ID_i=new_mat_ID_i, mat_ID_j=new_mat_ID_j, ele_ID=ele_ID)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()
        # Check length
        self._CheckL_CheckL()


class ForceBasedElement(MemberModel):
    """
    Class that handles the storage and manipulation of a force-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

    @param MemberModel: Parent abstract class.
    """
    def __init__(self, iNode_ID: int, jNode_ID: int, fiber_ID: int, geo_transf_ID: int,
        new_integration_ID = -1, Ip = 5, integration_type = "Lobatto", max_iter =
```

```
MAX_ITER_INTEGRATION, tol = TOL_INTEGRATION, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param fiber_ID (int): ID of the fiber section.
        @param geo_transf_ID (int): The geometric transformation (for more information, see OpenSeesPy documentation).
        @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
        @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
        @param integration_type (str, optional): Integration type. FOr more information, see OpenSeesPy documentation.
            Defaults to "Lobatto".
        @param max_iter (int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
        @param tol (float, optional): Tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer, if different from -1.
        @exception NegativeValue: Ip needs to be a positive integer bigger than 3, if different from -1.
        @exception NegativeValue: max_iter needs to be a positive integer.
        @exception NegativeValue: tol needs to be positive.
        @exception NegativeValue: ID needs to be a positive integer, if different from -1.
        """
        # Check
        if iNode_ID < 1: raise NegativeValue()
        if jNode_ID < 1: raise NegativeValue()
        if fiber_ID < 1: raise NegativeValue()
        if geo_transf_ID < 1: raise NegativeValue()
        if new_integration_ID != -1 and new_integration_ID < 1: raise NegativeValue()
        if Ip != -1 and Ip < 3: raise NegativeValue()
        if max_iter < 0: raise NegativeValue()
        if tol < 0: raise NegativeValue()
        if ele_ID != -1 and ele_ID < 1: raise NegativeValue()

        # Arguments
        self.iNode_IDiNode_ID = iNode_ID
        self.jNode_IDjNode_ID = jNode_ID
        self.fiber_IDfiber_ID = fiber_ID
        self.geo_transf_IDgeo_transf_ID = geo_transf_ID
        self.IpIp = Ip
        self.integration_typeintegration_type = integration_type
        self.max_itermax_iter = max_iter
        self.toltol = tol

        # Initialized the parameters that are dependent from others
        self.section_name_tagsection_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit(new_integration_ID, ele_ID)


    def ReInit(self, new_integration_ID, ele_ID = -1):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param new_integration_ID (int): ID of the integration technique.
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
        """
        # Precompute some members
        self.element_IDelement_ID = IDConvention(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID) if ele_ID == -1 else ele_ID

        # Arguments
        self.new_integration_IDnew_integration_ID = self.element_IDelement_ID if new_integration_ID == -1 else new_integration_ID

        # Members
        if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag = self.section_name_tagsection_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    # Methods
    def UpdateStoredData(self):
```

```
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "ForceBasedElement"], # Tag for differentiating different data
            ["element_ID", self.element_IDelement_ID],
            ["section_name_tag", self.section_name_tagsection_name_tag],
            ["Ip", self.IpIp],
            ["iNode_ID", self.iNode_IDiNode_ID],
            ["jNode_ID", self.jNode_IDjNode_ID],
            ["fiber_ID", self.fiber_IDfiber_ID],
            ["new_integration_ID", self.new_integration_IDnew_integration_ID],
            ["integration_type", self.integration_typeintegration_type],
            ["tol", self.toltol],
            ["max_iter", self.max_itermax_iter],
            ["tranf_ID", self.geo_transf_IDgeo_transf_ID],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self, plot = False, block = False):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        """
        print("")
        print("Requested info for ForceBasedElement member model, ID = {}".format(self.element_IDelement_ID))
        print("Fiber associated, ID = {} ".format(self.fiber_IDfiber_ID))
        print("Integration type '{}', ID = {}".format(self.integration_typeintegration_type, self.new_integration_IDnew_integration_ID))
        print("Section associated {} ".format(self.section_name_tagsection_name_tag))
        print("Number of integration points along the element Ip = {}, max iter = {}, tol = {}".format(self.IpIp, self.max_itermax_iter, self.toltol))
        print("Geometric transformation = {}".format(self.geo_transf_IDgeo_transf_ID))
        print("")

        if plot:
            if self.InitializedInitialized:
                plot_member(self.element_arrayelement_array, "ForceBased Element, ID = {}".format(self.element_IDelement_ID))
                if block:
                    plt.show()
            else:
                print("The ForceBasedElement is not initialized (element not created), ID = {}".format(self.element_IDelement_ID))


    def CreateMember(self):
        """
        Method that initialises the member by calling the OpenSeesPy commands through various functions.
        """
        self.element_arrayelement_array = [[self.element_IDelement_ID, self.iNode_IDiNode_ID, self.jNode_IDjNode_ID]]

        # Define integration type
        beamIntegration(self.integration_typeintegration_type, self.new_integration_IDnew_integration_ID, self.fiber_IDfiber_ID, self.IpIp)

        # Define element
        element('forceBeamColumn', self.element_IDelement_ID, self.iNode_IDiNode_ID, self.jNode_IDjNode_ID, self.geo_transf_IDgeo_transf_ID, self.new_integration_IDnew_integration_ID, '-iter', self.max_itermax_iter, self.toltol)

        # Update class
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


    def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
        """
        Implementation of the homonym abstract method.
        See parent class MemberModel for detailed information.
        """
        super().Record(self.element_IDelement_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec, time_rec=time_rec)


    def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
        """
        Implementation of the homonym abstract method.
        See parent class MemberModel for detailed information.
```

```
        """
        super().RecordNodeDef(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID, name_txt, data_dir, time_rec=time_rec)


class ForceBasedElementFibersRectRCRectShape(ForceBasedElement):
    """
    Class that is the children of ForceBasedElement and combine the class FibersRectRCRectShape (fiber section) to retrieve the information needed.

    @param ForceBasedElement: Parent class.
    """
    def __init__(self, iNode_ID: int, jNode_ID: int, fiber: FibersRectRCRectShape, geo_transf_ID: int,
        new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION, tol=TOL_INTEGRATION, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param fiber (FibersRectRCRectShape): FibersRectRCRectShape fiber section object.
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
        @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
        @param integration_type (str, optional): Integration type. FOr more information, see OpenSeesPy documentation.
            Defaults to "Lobatto".
        @param max_iter (int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
        @param tol (float, optional): Tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
        """
        self.sectionsection = deepcopy(fiber.section)
        super().__init__(iNode_ID, jNode_ID, fiber.ID, geo_transf_ID,
            new_integration_ID=new_integration_ID, Ip=Ip, integration_type=integration_type, max_iter=max_iter, tol=tol, ele_ID= ele_ID)
        self.section_name_tagsection_name_tagsection_name_tag = self.sectionsection.name_tag
        self.UpdateStoredDataUpdateStoredData()
        # Check length
        self._CheckL_CheckL()


class ForceBasedElementFibersCircRCCircShape(ForceBasedElement):
    """
    Class that is the children of ForceBasedElement and combine the class FibersCircRCCircShape (fiber section) to retrieve the information needed.

    @param ForceBasedElement: Parent class.
    """
    def __init__(self, iNode_ID: int, jNode_ID: int, fiber: FibersCircRCCircShape, geo_transf_ID: int,
        new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION, tol=TOL_INTEGRATION, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param fiber (FibersCircRCCircShape): FibersCircRCCircShape fiber section object.
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
        @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
        @param integration_type (str, optional): Integration type. FOr more information, see OpenSeesPy documentation.
            Defaults to "Lobatto".
        @param max_iter (int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
        @param tol (float, optional): Tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
        """
        self.sectionsection = deepcopy(fiber.section)
        super().__init__(iNode_ID, jNode_ID, fiber.ID, geo_transf_ID,
            new_integration_ID=new_integration_ID, Ip=Ip, integration_type=integration_type, max_iter=max_iter, tol=tol, ele_ID=ele_ID)
        self.section_name_tagsection_name_tagsection_name_tag = self.sectionsection.name_tag
        self.UpdateStoredDataUpdateStoredData()
        # Check length
        self._CheckL_CheckL()


```

```
class ForceBasedElementFibersIShapeSteelIShape(ForceBasedElement):
    """
    Class that is the children of ForceBasedElement and combine the class FibersIShapeSteelIShape (fiber section) to retrieve the information needed.

    @param ForceBasedElement: Parent class.
    """
    def __init__(self, iNode_ID: int, jNode_ID: int, fiber: FibersIShapeSteelIShape, geo_transf_ID: int,
        new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION, tol=TOL_INTEGRATION, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param fiber (FibersIShapeSteelIShape): FibersIShapeSteelIShape fiber section object.
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
        @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
        @param integration_type (str, optional): Integration type. FOr more information, see OpenSeesPy documentation.
            Defaults to "Lobatto".
        @param max_iter (int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
        @param tol (float, optional): Tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
        """
        self.sectionsection = deepcopy(fiber.section)
        super().__init__(iNode_ID, jNode_ID, fiber.ID, geo_transf_ID,
            new_integration_ID=new_integration_ID, Ip=Ip, integration_type=integration_type, max_iter=max_iter, tol=tol, ele_ID=ele_ID)
        self.section_name_tagsection_name_tagsection_name_tag = self.sectionsection.name_tag
        self.UpdateStoredDataUpdateStoredData()
        # Check length
        self._CheckL_CheckL()


class GIFBElement(MemberModel):
    """
    Class that handles the storage and manipulation of a Gradient-Inelastic Flexibility-based element’s information
        (mechanical and geometrical parameters, etc) and the initialisation in the model.
    The integration technique is Simpson. For more information, see Sideris and Salehi 2016, 2017 and 2020.

    @param MemberModel: Parent abstract class.
    """
    def __init__(self, iNode_ID: int, jNode_ID: int, fiber_ID: int, D_bars, fy, geo_transf_ID: int,
        lambda_i = -1, lambda_j = -1, Lp = -1, Ip = -1, new_integration_ID = -1,
        min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param fiber_ID (int): ID of the fiber section.
        @param D_bars (float): Diameter of the vertical reinforcing bars.
        @param fy (float): Yield stress of the reinforcing bars.
        @param geo_transf_ID (int): The geometric transformation (for more information, see OpenSeesPy documentation).
        @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge).
            Defaults to -1, e.g. plastic hinge in the end i.
        @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge).
            Defaults to -1, e.g. plastic hinge in the end j.
        @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
        @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
        @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
        @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
        @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
        @param max_iter (int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer.
```

```
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: D_bars needs to be positive.
        @exception NegativeValue: fy needs to be positive.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: lambda_i needs to be positive.
        @exception NegativeValue: lambda_j needs to be positive.
        @exception NegativeValue: No plastic length defined.
        @exception NegativeValue: Lp needs to be positive, if different from -1.
        @exception NegativeValue: Ip needs to be a positive integer bigger than 3, if different from -1.
        @exception NegativeValue: ID needs to be a positive integer.
        @exception NegativeValue: min_tol needs to be positive.
        @exception NegativeValue: max_tol needs to be positive.
        @exception NegativeValue: max_iter needs to be a positive integer.
        @exception NegativeValue: ID needs to be a positive integer, if different from -1.
        """
        # Check
        if iNode_ID < 1: raise NegativeValue()
        if jNode_ID < 1: raise NegativeValue()
        if fiber_ID < 1: raise NegativeValue()
        if D_bars < 0: raise NegativeValue()
        if fy < 0: raise NegativeValue()
        if geo_transf_ID < 1: raise NegativeValue()
        if lambda_i != -1 and lambda_i < 0: raise NegativeValue()
        if lambda_j != -1 and lambda_j < 0: raise NegativeValue()
        if lambda_i == 0 and lambda_j == 0: print("!!!!!!! WARNING !!!!!!! No plastic length defined for element ID = {}".format(IDConvention(iNode_ID, jNode_ID)))
        if Lp != -1 and Lp < 0: raise NegativeValue()
        if Ip != -1 and Ip < 3: raise NegativeValue()
        if new_integration_ID != -1 and new_integration_ID < 1: raise NegativeValue()
        if min_tol < 0: raise NegativeValue()
        if max_tol < 0: raise NegativeValue()
        if max_iter < 0: raise NegativeValue()
        if ele_ID != -1 and ele_ID < 0: raise NegativeValue()

        # Arguments
        self.iNode_IDiNode_ID = iNode_ID
        self.jNode_IDjNode_ID = jNode_ID
        self.D_barsD_bars = D_bars
        self.fyfy = fy
        self.geo_transf_IDgeo_transf_ID = geo_transf_ID
        self.fiber_IDfiber_ID = fiber_ID
        self.min_tolmin_tol = min_tol
        self.max_tolmax_tol = max_tol
        self.max_itermax_iter = max_iter

        # Initialized the parameters that are dependent from others
        self.section_name_tagsection_name_tag = "None"
        self.InitializedInitialized = False
        self.ReInitReInit(lambda_i, lambda_j, Lp, Ip, new_integration_ID, ele_ID)

    def ReInit(self, lambda_i = -1, lambda_j = -1, Lp = -1, Ip = 5, new_integration_ID = -1, ele_ID = -1):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge).
            Defaults to -1, e.g. plastic hinge in the end i.
        @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge).
            Defaults to -1, e.g. plastic hinge in the end j.
        @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed here.
        @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
        @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
        """
        # Precompute some members
        iNode = np.array(nodeCoord(self.iNode_IDiNode_ID))
        jNode = np.array(nodeCoord(self.jNode_IDjNode_ID))
        self.LL = np.linalg.norm(iNode-jNode)
        self.element_IDelement_ID = IDConvention(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID) if ele_ID == -1 else ele_ID

        # Arguments
        self.LpLp = self.ComputeLpComputeLp() if Lp == -1 else Lp
        self.IpIp = self.ComputeIpComputeIp() if Ip == -1 else Ip
        self.lambda_ilambda_i = self.LpLp/self.LL if lambda_i == -1 else lambda_i
        self.lambda_jlambda_j = self.LpLp/self.LL if lambda_j == -1 else lambda_j
        self.new_integration_IDnew_integration_ID = self.element_IDelement_ID if new_integration_ID == -1 else new_integration_ID

        # Members
        if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
```

```
self.section_name_tagsection_name_tag + " (modified)"

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    # Methods
    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "GIFBElement"], # Tag for differentiating different data
            ["element_ID", self.element_IDelement_ID],
            ["section_name_tag", self.section_name_tagsection_name_tag],
            ["L", self.LL],
            ["D_bars", self.D_barsD_bars],
            ["fy", self.fyfy],
            ["Lp", self.LpLp],
            ["Ip", self.IpIp],
            ["iNode_ID", self.iNode_IDiNode_ID],
            ["lambda_i", self.lambda_ilambda_i],
            ["jNode_ID", self.jNode_IDjNode_ID],
            ["lambda_j", self.lambda_jlambda_j],
            ["fiber_ID", self.fiber_IDfiber_ID],
            ["new_integration_ID", self.new_integration_IDnew_integration_ID],
            ["min_tol", self.min_tolmin_tol],
            ["max_tol", self.max_tolmax_tol],
            ["max_iter", self.max_itermax_iter],
            ["tranf_ID", self.geo_transf_IDgeo_transf_ID],
            ["Initialized", self.InitializedInitialized]]


    def ShowInfo(self, plot = False, block = False):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param plot (bool, optional): Option to show the plot of the material model. Defaults to False.
        @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
        """
        print("")
        print("Requested info for GIFBElement member model, ID = {}".format(self.element_IDelement_ID))
        print("Fiber associated, ID = {} ".format(self.fiber_IDfiber_ID))
        print("Integration type 'Simpson', ID = {}".format(self.new_integration_IDnew_integration_ID))
        print("Section associated {} ".format(self.section_name_tagsection_name_tag))
        print("Length L = {} m".format(self.LL/m_unit))
        print("Diameter of the reinforcing bars D_bars = {} mm2".format(self.D_barsD_bars/mm2_unit))
        print("Reinforcing bar steel strength fy = {} MPa".format(self.fyfy/MPa_unit))
        print("Plastic length Lp = {} mm".format(self.LpLp/mm_unit))
        print("Number of integration points along the element Ip = {}, max iter = {}, (min, max tol) = ({},{})".format(self.IpIp, self.max_itermax_iter, self.min_tolmin_tol, self.max_tolmax_tol))
        print("Lambda_i = {} and lambda_j = {}".format(self.lambda_ilambda_i, self.lambda_jlambda_j))
        print("Geometric transformation = {}".format(self.geo_transf_IDgeo_transf_ID))
        print("")

        if plot:
            if self.InitializedInitialized:
                plot_member(self.element_arrayelement_array, "GIFB Element, ID = {}".format(self.element_IDelement_ID))
                if block:
                    plt.show()
            else:
                print("The GIFBElement is not initialized (element not created), ID = {}".format(self.element_IDelement_ID))


    def CreateMember(self):
        """
        Method that initialises the member by calling the OpenSeesPy commands through various functions.
        """
        self.element_arrayelement_array = [[self.element_IDelement_ID, self.iNode_IDiNode_ID, self.jNode_IDjNode_ID]]

        # Define integration type
        beamIntegration('Simpson', self.new_integration_IDnew_integration_ID, self.fiber_IDfiber_ID, self.IpIp)

        # Define element TODO: Dr. Salehi: lambda useless
        element('gradientInelasticBeamColumn', self.element_IDelement_ID, self.iNode_IDiNode_ID, self.jNode_IDjNode_ID, self.geo_transf_IDgeo_transf_ID,
            self.new_integration_IDnew_integration_ID, self.lambda_ilambda_i, self.lambda_jlambda_j, self.LpLp, '-iter', self.max_itermax_iter, self.min_tolmin_tol, self.max_tolmax_tol)
```

```

        # Update class
        self.InitializedInitialized = True
        self.UpdateStoredDataUpdateStoredData()


    def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
        """
        Implementation of the homonym abstract method.
        See parent class MemberModel for detailed information.
        """
        super().Record(self.element_IDelement_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec, time_rec=time_rec)


    def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
        """
        Implementation of the homonym abstract method.
        See parent class MemberModel for detailed information.
        """
        super().RecordNodeDef(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID, name_txt, data_dir, time_rec=time_rec)


    def ComputeLp(self):
        """
        Method that computes the plastic length using Paulay 1992.

        @returns double: Plastic length
        """
        return (0.08*self.LL/m_unit + 0.022*self.D_barsD_bars/m_unit*self.fyfy/MPa_unit)*m_unit


    def ComputeIp(self):
        """
        Compute the number of integration points with equal distance along the element. For more information, see Salehi and Sideris 2020.

        @returns int: Number of integration points
        """
        tmp = math.ceil(1.5*self.LL/self.LpLp + 1)
        if (tmp % 2) == 0:
            return tmp + 1
        else:
            return tmp


class GIFBElementRCRectShape(GIFBElement):
    """
    Class that is the children of GIFBElement and combine the class RCRectShape (section) to retrieve the information needed.

    @param GIFBElement: Parent class.
    """
    def __init__(self, iNode_ID: int, jNode_ID: int, fiber_ID: int, section: RCRectShape, geo_transf_ID: int,
        lambda_i=-1, lambda_j=-1, Lp=-1, Ip=-1, new_integration_ID=-1,
        min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param fiber_ID (int): ID of the fiber section.
        @param section (RCRectShape): RCRectShape section object.
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge).
            Defaults to -1, e.g. plastic hinge in the end i.
        @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge).
            Defaults to -1, e.g. plastic hinge in the end j.
        @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
        @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
        @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
        @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
        @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
        @param max_iter (int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
        """
```

```
        self.sectionsection = deepcopy(section)
        super().__init__(iNode_ID, jNode_ID, fiber_ID, section.D_bars, section.fy, geo_transf_ID,
            lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
            min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()
        # Check length
        self._CheckL_CheckL()


class GIFBElementFibersRectRCRectShape(GIFBElement):
    """
    Class that is the children of GIFBElement and combine the class FibersRectRCRectShape (fiber section) to retrieve the information needed.

    @param GIFBElement: Parent class.
    """
    def __init__(self, iNode_ID: int, jNode_ID: int, fib: FibersRectRCRectShape, geo_transf_ID: int,
        lambda_i=-1, lambda_j=-1, Lp=-1, Ip=-1, new_integration_ID=-1,
        min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param fib (FibersRectRCRectShape): FibersRectRCRectShape fiber section object.
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge).
            Defaults to -1, e.g. plastic hinge in the end i.
        @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge).
            Defaults to -1, e.g. plastic hinge in the end j.
        @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
        @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
        @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
        @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
        @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
        @param max_iter (int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
        """
        self.sectionsection = deepcopy(fib.section)
        super().__init__(iNode_ID, jNode_ID, fib.ID, self.sectionsection.D_bars, self.sectionsection.fy, geo_transf_ID,
            lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
            min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
        self.section_name_tagsection_name_tagsection_name_tag = self.sectionsection.name_tag
        self.UpdateStoredDataUpdateStoredData()
        # Check length
        self._CheckL_CheckL()


class GIFBElementRCCircShape(GIFBElement):
    """
    Class that is the children of GIFBElement and combine the class RCCircShape (section) to retrieve the information needed.

    @param GIFBElement: Parent class.
    """
    def __init__(self, iNode_ID: int, jNode_ID: int, fiber_ID: int, section: RCCircShape, geo_transf_ID: int,
        lambda_i=-1, lambda_j=-1, Lp=-1, Ip=-1, new_integration_ID=-1,
        min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param fiber_ID (int): ID of the fiber section.
        @param section (RCCircShape): RCCircShape section object.
        @param geo_transf_ID (int): The geometric transformation (for more information, see OpenSeesPy documentation).
        @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge).
            Defaults to -1, e.g. plastic hinge in the end i.
        @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge).
            Defaults to -1, e.g. plastic hinge in the end j.
        @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
```

```
        @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
        @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
        @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
        @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
        @param max_iter (int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
        """
        self.sectionsection = deepcopy(section)
        super().__init__(iNode_ID, jNode_ID, fiber_ID, section.D_bars, section.fy, geo_transf_ID,
            lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
            min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
        self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
        self.UpdateStoredDataUpdateStoredData()
        # Check length
        self._CheckL_CheckL()


class GIFBElementFibersCircRCCircShape(GIFBElement):
    """
    Class that is the children of GIFBElement and combine the class FibersCircRCCircShape (fiber section) to retrieve the information needed.

    @param GIFBElement: Parent class.
    """
    def __init__(self, iNode_ID: int, jNode_ID: int, fib: FibersCircRCCircShape, geo_transf_ID: int,
        lambda_i=-1, lambda_j=-1, Lp=-1, Ip=-1, new_integration_ID=-1,
        min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION, ele_ID = -1):
        """
        Constructor of the class.

        @param iNode_ID (int): ID of the first end node.
        @param jNode_ID (int): ID of the second end node.
        @param fib (FibersCircRCCircShape): FibersCircRCCircShape fiber section object.
        @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy documentation).
        @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge).
            Defaults to -1, e.g. plastic hinge in the end i.
        @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge).
            Defaults to -1, e.g. plastic hinge in the end j.
        @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
        @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
        @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1, e.g. computed in ReInit().
        @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION (Units).
        @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
        @param max_iter (int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION (Units).
        @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.
        """
        self.sectionsection = deepcopy(fib.section)
        super().__init__(iNode_ID, jNode_ID, fib.ID, self.sectionsection.D_bars, self.sectionsection.fy, geo_transf_ID,
            lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
            min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
        self.section_name_tagsection_name_tagsection_name_tag = self.sectionsection.name_tag
        self.UpdateStoredDataUpdateStoredData()
        # Check length
        self._CheckL_CheckL()
```

8.21 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/README.md File Reference

8.22 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Section.py File Reference

Classes

- class RCCircShape

  Class that stores funcions, geometric and mechanical properties of RC circular shape profile.
- class RCRectShape

  Class that stores funcions, geometric and mechanical properties of RC rectangular shape profile.
- class RCSquareShape

  Class that is the children of RCRectShape and cover the specific case of square RC sections.
- class Section

  Parent abstract class for the storage and manipulation of a section's information (mechanical and geometrical parameters, etc).
- class SteelIShape

  Class that stores funcions, geometric and mechanical properties of a steel double symmetric I-shape profile.

Namespaces

- namespace Section

  Module for the section (steel I shape profiles, RC circular/square/rectangular sections).

Functions

- def ComputeACircle (D)

  Function that computes the area of one circle (reinforcing bar or hoop).
- def ComputeRho (A, nr, A_tot)

  Compute the ratio of area of a reinforcement to area of a section.

8.23 Section.py

Go to the documentation of this file.

```
"""
Module for the section (steel I shape profiles, RC circular/square/rectangular sections).
Carmine Schipani, 2021
"""

import numpy as np
import math
from copy import copy, deepcopy
from OpenSeesPyAssistant.DataManagement import *
from OpenSeesPyAssistant.ErrorHandling import *
from OpenSeesPyAssistant.Units import *

class Section(DataManagement):
    """
    Parent abstract class for the storage and manipulation of a section's information (mechanical and geometrical parameters, etc).

    @param DataManagement: Parent abstract class.
    """
```

```
    pass


class SteelIShape(Section):
    """
    Class that stores funcions, geometric and mechanical properties of a steel double symmetric I-shape profile.
    The parameter 'n' is used as global throughout the SteelIShape sections to optimise the program (given the fact that is constant everytime).

    @param Section: Parent abstract class.
    """
    global n
    n = 10.0

    def __init__(self, Type: str, d, bf, tf, tw, L, r, E, Fy, Fy_web = -1, name_tag = "Not Defined"):
        """
        The conctructor of the class.

        @param Type (str): Type of the section. It can be 'Col' for column or 'Beam' for beams.
        @param d (float): Depth of the section.
        @param bf (float): Flange's width of the section
        @param tf (float): Flange's thickness of the section
        @param tw (float): Web's thickness of the section
        @param L (float): Effective length of the element associated with this section.
            If the panel zone is present, exclude its dimension.
        @param r (float): Radius of the weld fillets of the section.
        @param E (float): Young modulus of the section.
        @param Fy (float): Yield strength of the flange of the section. Used as the yield strength of the entire section.
        @param Fy_web (float, optional): Yield strength of the web of the section. Used for panel zone associated to this section.
            Defaults to -1, e.g. computed in __init__() as equal to Fy.
        @param name_tag (str, optional): Name TAG of the section. Defaults to "Not Defined".

        @exception WrongArgument: Type needs to be 'Col' or 'Beam'.
        @exception NegativeValue: d needs to be positive.
        @exception NegativeValue: bf needs to be positive.
        @exception NegativeValue: tf needs to be positive.
        @exception NegativeValue: tw needs to be positive.
        @exception NegativeValue: L needs to be positive.
        @exception NegativeValue: r needs to be positive.
        @exception NegativeValue: E needs to be positive.
        @exception NegativeValue: Fy needs to be positive.
        @exception NegativeValue: Fy_web needs to be positive if different from -1.
        @exception InconsistentGeometry: tw should be smaller than bf.
        @exception InconsistentGeometry: tf needs to be smaller than half of d
        @exception InconsistentGeometry: r should be less than half bf and d
        """
        # Check
        if Type != "Beam" and Type != "Col": raise WrongArgument()
        if d < 0: raise NegativeValue()
        if bf < 0: raise NegativeValue()
        if tf < 0: raise NegativeValue()
        if tw < 0: raise NegativeValue()
        if L < 0: raise NegativeValue()
        if r < 0: raise NegativeValue()
        if E < 0: raise NegativeValue()
        if Fy < 0: raise NegativeValue()
        if Fy_web != -1 and Fy_web < 0: raise NegativeValue()
        if tw > bf: raise InconsistentGeometry()
        if tf > d/2: raise InconsistentGeometry()
        if r > bf/2 or r > d/2: raise InconsistentGeometry()

        # Arguments
        self.TypeType = Type
        self.dd = d
        self.bfbf = bf
        self.tftf = tf
        self.twtw = tw
        self.LL = L
        self.rr = r
        self.EE = E
        self.FyFy = Fy
        self.Fy_webFy_web = Fy if Fy_web == -1 else Fy_web
        self.name_tagname_tag = name_tag

        # Initialized the parameters that are dependent from others
        self.ReInitReInit()

    def ReInit(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        # Member
        self.h_1h_1 = self.dd - 2.0*self.rr -2.0*self.tftf
```

```
        self.AA = self.ComputeAComputeA()
        self.NplNpl = self.AA*self.FyFy
        self.IyIy = self.ComputeIyComputeIy()
        self.IzIz = self.ComputeIzComputeIz()
        self.WplyWply = self.ComputeWplyComputeWply()
        self.WplzWplz = self.ComputeWplzComputeWplz()
        self.MyMy = self.FyFy*self.WplyWply
        self.Iy_modIy_mod = self.IyIy*(n + 1.0)/n
        self.iziz = self.Compute_izCompute_iz()
        self.iyiy = self.Compute_iyCompute_iy()

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()

    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "SteelIShape"], # Tag for differentiating different data
            ["name_tag", self.name_tagname_tag],
            ["Type", self.TypeType],
            ["d", self.dd],
            ["bf", self.bfbf],
            ["tf", self.tftf],
            ["tw", self.twtw],
            ["L", self.LL],
            ["r", self.rr],
            ["h_1", self.h_1h_1],
            ["E", self.EE],
            ["Fy", self.FyFy],
            ["Fy_web", self.Fy_webFy_web],
            ["A", self.AA],
            ["Iy", self.IyIy],
            ["Iz", self.IzIz],
            ["Wply", self.WplyWply],
            ["Wplz", self.WplzWplz],
            ["Iy_mod", self.Iy_modIy_mod],
            ["iy", self.iyiy],
            ["iz", self.iziz],
            ["Npl", self.NplNpl],
            ["My", self.MyMy]]

    def ShowInfo(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        print("")
        print("Requested info for steel I shape section of type = {} and name tag =
{}".format(self.TypeType, self.name_tagname_tag))
        print("d = {} mm".format(self.dd/mm_unit))
        print("Fy = {} MPa".format(self.FyFy/MPa_unit))
        print("Fy web = {} MPa".format(self.Fy_webFy_web/MPa_unit))
        print("E = {} GPa".format(self.EE/GPa_unit))
        print("h_1 = {} mm".format(self.h_1h_1/mm_unit))
        print("A = {} mm2".format(self.AA/mm2_unit))
        print("Iy = {} mm4".format(self.IyIy/mm4_unit))
        print("Iz = {} mm4".format(self.IzIz/mm4_unit))
        print("Wply = {} mm3".format(self.WplyWply/mm3_unit))
        print("Wplz = {} mm3".format(self.WplzWplz/mm3_unit))
        print("Iy_mod = {} mm4".format(self.Iy_modIy_mod/mm4_unit))
        print("iy = {} mm".format(self.iyiy/mm_unit))
        print("iz = {} mm".format(self.iziz/mm_unit))
        print("My = {} kNm".format(self.MyMy/kNm_unit))
        print("Npl = {} kN".format(self.NplNpl/kN_unit))
        print("")


    def ComputeA(self):
        """
        Compute the area of a double symmetric I-profile section with fillets.

        @returns float: Area of the I shape section (with fillets included)
        """
        #  d :     The depth
        #  bf :    The flange's width
        #  tf :    The flange's thickness
        #  tw :    The web's thickness
        #  r :     The weld fillet radius

        # without fillets bf*tf*2 + tw*(d-2*tf)
        return 2.0*self.bfbf*self.tftf+self.twtw*(self.dd-2.0*self.tftf)+0.8584*self.rr**2

    def ComputeIy(self):
        """
        Compute the moment of inertia of a double symmetric I-profile section, with respect to its
```

```
strong axis with fillets.

        @returns float: The moment of inertia with respect to the strong axis.
        """
        #  d :     The depth
        #  bf :    The flange's width
        #  tf :    The flange's thickness
        #  tw :    The web's thickness
        #  r :     The weld fillet radius

        # without fillets: bf*tf/2*(d-tf)**2 + bf*tf**3/6 + (d-tf*2)**3*tf/12
        return
 (self.bfbf*self.dd**3.0-(self.bfbf-self.twtw)*(self.dd-2.0*self.tftf)**3)/12.0+0.8584*self.rr**2*(0.5*self.dd-self.tft

    def ComputeIz(self):
        """
        Compute the moment of inertia of a double symmetric I-profile section, with respect to its
weak axis with fillets.

        @returns float: The moment of inertia with respect to the weak axis.
        """
        #  d :     The depth
        #  bf :    The flange's width
        #  tf :    The flange's thickness
        #  tw :    The web's thickness
        #  r :     The weld fillet radius

        return
 (self.tftf*self.bfbf**3)/6.0+((self.dd-2.0*self.tftf)*self.twtw**3)/12.0+0.8584*self.rr**2*(0.5*self.twtw+0.2234*self.r

    def ComputeWply(self):
        """
        Compute the plastic modulus of a double symmetric I-profile section, with respect to its
strong axis with fillets.

        @returns float: The plastic modulus with respect to the strong axis.
        """
        #  d :     The depth
        #  bf :    The flange's width
        #  tf :    The flange's thickness
        #  tw :    The web's thickness
        #  r :     The weld fillet radius

        return
self.bfbf*self.tftf*(self.dd-self.tftf)+(self.dd-2.0*self.tftf)**2.0*(self.twtw/4.0)+0.4292*self.rr**2*(self.dd-2.0*se

    def ComputeWplz(self):
        """
        Compute the plastic modulus of a double symmetric I-profile section, with respect to its weak
axis with fillets.

        @returns float: The plastic modulus with respect to the weak axis.
        """
        #  d :     The depth
        #  bf :    The flange's width
        #  tf :    The flange's thickness
        #  tw :    The web's thickness
        #  r :     The weld fillet radius
        return
 (self.tftf*self.bfbf**2)/2+(self.dd-2.0*self.tftf)*(self.twtw**2/4.0)+0.4292*self.rr**2*(self.twtw+0.4467*self.rr)

    def Compute_iy(self):
        """
        Compute the gyration radius with respect to the strong axis.

        @returns float: The gyration radius with respect to the strong axis.
        """
        #  Iy :    The second moment of inertia with respect to thte strong axis
        #  A :     The area

        return math.sqrt(self.IyIy/self.AA)

    def Compute_iz(self):
        """
        Compute the gyration radius with respect to the weak axis.

        @returns float: The gyration radius with respect to the weak axis.
        """
        #  Iz :    The second moment of inertia with respect to thte weak axis
        #  A :     The area

        return math.sqrt(self.IzIz/self.AA)


class RCRectShape(Section):
    """
    Class that stores funcions, geometric and mechanical properties of RC rectangular shape profile.
```

```
    Note that for the validity of the formulas, at least one bar per corner and at least one hoop closed (with 135 degress possibly).

    @param Section: Parent abstract class.
    """
    def __init__(self, b, d, L, e, fc, D_bars, bars_position_x: np.ndarray, bars_ranges_position_y: np.ndarray, fy, Ey,
        D_hoops, s, fs, Es, name_tag = "Not Defined", rho_s_x = -1, rho_s_y = -1, Ec = -1):
        """
        The conctructor of the class.

        @param b (float): Width of the section.
        @param d (float): Depth of the section.
        @param L (float): Effective length of the element associated with this section.
            If the panel zone is present, exclude its dimension.
        @param e (float): Concrete cover.
        @param fc (float): Unconfined concrete compressive strength (cylinder test).
        @param D_bars (float): Diameter of the reinforcing bars.
        @param bars_position_x (np.ndarray): Array with a range of aligned vertical reinforcing bars for each row in x direction.
            Distances from border to bar centerline, bar to bar centerlines and
            finally bar centerline to border in the x direction (aligned).
            Starting from the left to right, from the top range to the bottom one.
            The number of bars for each range can vary; in this case, add this argument when defining the array " dtype = object".
        @param bars_ranges_position_y (np.ndarray): Array of dimension 1 with the position or spacing in y of the ranges in bars_position_x.
            Distances from border to range centerlines, range to range centerlines and
            finally range centerline to border in the y direction.
            Starting from the top range to the bottom one.
        @param fy (float): Yield stress for reinforcing bars.
        @param Ey (float): Young modulus for reinforcing bars.
        @param D_hoops (float): Diameter of the hoops.
        @param s (float): Centerline distance for the hoops.
        @param fs (float): Yield stress for the hoops.
        @param Es (float): Young modulus for the hoops
        @param name_tag (str, optional): A nametag for the section. Defaults to "Not Defined".
        @param rho_s_x (float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the x direction.
            Defaults to -1, e.g. computed in __init__()  and ReInit() assuming one range of hoops.
        @param rho_s_y (float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the y direction.
            Defaults to -1, e.g. computed in __init__()  and ReInit() assuming one range of hoops.
        @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in __init__() and ReInit().

        @exception NegativeValue: b needs to be positive.
        @exception NegativeValue: d needs to be positive.
        @exception NegativeValue: L needs to be positive.
        @exception NegativeValue: e needs to be positive.
        @exception PositiveValue: fc needs to be negative.
        @exception NegativeValue: D_bars needs to be positive.
        @exception NegativeValue: fy needs to be positive.
        @exception NegativeValue: Ey needs to be positive.
        @exception NegativeValue: D_hoops needs to be positive.
        @exception NegativeValue: s needs to be positive.
        @exception NegativeValue: fs needs to be positive.
        @exception NegativeValue: Es needs to be positive.
        @exception NegativeValue: rho_s_x needs to be positive if different from -1.
        @exception NegativeValue: rho_s_y needs to be positive if different from -1.
        @exception NegativeValue: Ec needs to be positive if different from -1.
        @exception WrongDimension: Number of lists in the list bars_position_x needs to be the same of the length of bars_ranges_position_y - 1.
        @exception InconsistentGeometry: The sum of the distances for each list in bars_position_x should be equal to the section's width (tol = 5 mm).
        @exception InconsistentGeometry: The sum of the distances in bars_ranges_position_y should be equal to the section's depth (tol = 5 mm).
        @exception InconsistentGeometry: e should be smaller than half the depth and the width of the section.
        """
        # Check
        if b < 0: raise NegativeValue()
        if d < 0: raise NegativeValue()
        if L < 0: raise NegativeValue()
        if e < 0: raise NegativeValue()
        if fc > 0: raise PositiveValue()
        if D_bars < 0: raise NegativeValue()
        if fy < 0: raise NegativeValue()
        if Ey < 0: raise NegativeValue()
        if D_hoops < 0: raise NegativeValue()
        if s < 0: raise NegativeValue()
        if fs < 0: raise NegativeValue()
        if Es < 0: raise NegativeValue()
        if rho_s_x != -1 and rho_s_x < 0: raise NegativeValue()
        if rho_s_y != -1 and rho_s_y < 0: raise NegativeValue()
        if Ec != -1 and Ec < 0: raise NegativeValue()
        if np.size(bars_position_x) != np.size(bars_ranges_position_y)-1: raise WrongDimension()
```

```
        geometry_tol = 5*mm_unit
        for bars in bars_position_x:
            if abs(np.sum(bars) - b) > geometry_tol: raise InconsistentGeometry()
        if abs(np.sum(bars_ranges_position_y)-d) > geometry_tol: raise InconsistentGeometry()
        if e > b/2 or e > d/2: raise InconsistentGeometry()
        warning_min_bars = "!!!!!!! WARNING !!!!!!! The hypothesis of one bar per corner (aligned) is not fullfilled."
        if len(bars_position_x) < 2:
            print(warning_min_bars)
        elif len(bars_position_x[0]) < 3 or len(bars_position_x[-1]) < 3:
            print(warning_min_bars)

        # Arguments
        self.bb = b
        self.dd = d
        self.LL = L
        self.ee = e
        self.fcfc = fc
        self.D_barsD_bars = D_bars
        self.bars_position_xbars_position_x = deepcopy(bars_position_x)
        self.bars_ranges_position_ybars_ranges_position_y = copy(bars_ranges_position_y)
        self.fyfy = fy
        self.EyEy = Ey
        self.D_hoopsD_hoops = D_hoops
        self.ss = s
        self.fsfs = fs
        self.EsEs = Es
        self.name_tagname_tag = name_tag

        # Initialized the parameters that are dependent from others
        self.ReInitReInit(rho_s_x, rho_s_y, Ec)


    def ReInit(self, rho_s_x = -1, rho_s_y = -1, Ec = -1):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param rho_s_x (float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the x direction.
            Defaults to -1, e.g. computed assuming one range of hoops.
        @param rho_s_y (float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the y direction.
            Defaults to -1, e.g. computed assuming one range of hoops.
        @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed according to Mander et Al. 1988.
        """
        # Precompute some members
        self.cl_hoopscl_hoops = self.ee + self.D_hoopsD_hoops/2.0 # centerline distance from the border of the extreme confining hoops
        self.cl_barscl_bars = self.ee + self.D_barsD_bars/2.0 + self.D_hoopsD_hoops # centerline distance from the border of the corner bars
        self.bcbc = self.bb - self.cl_hoopscl_hoops*2
        self.dcdc = self.dd - self.cl_hoopscl_hoops*2
        self.AsAs = ComputeACircle(self.D_hoopsD_hoops)

        # Arguments
        self.rho_s_xrho_s_x = 2.0*ComputeRho(self.AsAs, 1, self.bcbc*self.ss) if rho_s_x == -1 else rho_s_x
        self.rho_s_yrho_s_y = 2.0*ComputeRho(self.AsAs, 1, self.dcdc*self.ss) if rho_s_y == -1 else rho_s_y
        self.EcEc = self.ComputeEcComputeEc() if Ec == -1 else Ec

        # Members
        self.nr_barsnr_bars = self.ComputeNrBarsComputeNrBars()
        self.AA = self.ComputeAComputeA()
        self.AcAc = self.ComputeAcComputeAc()
        self.AyAy = ComputeACircle(self.D_barsD_bars)
        self.rho_barsrho_bars = ComputeRho(self.AyAy, self.nr_barsnr_bars, self.AA)
        self.IyIy = self.ComputeIyComputeIy()
        self.IzIz = self.ComputeIzComputeIz()

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "RCRectShape"], # Tag for differentiating different data
            ["name_tag", self.name_tagname_tag],
            ["b", self.bb],
            ["d", self.dd],
            ["bc", self.bcbc],
            ["dc", self.dcdc],
```

```
            ["L", self.LL],
            ["e", self.ee],
            ["A", self.AA],
            ["Ac", self.AcAc],
            ["Iy", self.IyIy],
            ["Iz", self.IzIz],
            ["fc", self.fcfc],
            ["Ec", self.EcEc],
            ["D_bars", self.D_barsD_bars],
            ["nr_bars", self.nr_barsnr_bars],
            ["Ay", self.AyAy],
            ["bars_position_x", self.bars_position_xbars_position_x],
            ["bars_ranges_position_y", self.bars_ranges_position_ybars_ranges_position_y],
            ["rho_bars", self.rho_barsrho_bars],
            ["cl_bars", self.cl_barscl_bars],
            ["fy", self.fyfy],
            ["Ey", self.EyEy],
            ["D_hoops", self.D_hoopsD_hoops],
            ["s", self.ss],
            ["As", self.AsAs],
            ["rho_s_x", self.rho_s_xrho_s_x],
            ["rho_s_y", self.rho_s_yrho_s_y],
            ["cl_hoops", self.cl_hoopscl_hoops],
            ["fs", self.fsfs],
            ["Es", self.EsEs]]


    def ShowInfo(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        print("")
        print("Requested info for RC rectangular section of name tag = {}".format(self.name_tagname_tag))
        print("Width of the section b = {} mm".format(self.bb/mm_unit))
        print("Depth of the section d = {} mm".format(self.dd/mm_unit))
        print("Concrete cover e = {} mm".format(self.ee/mm_unit))
        print("Concrete area A = {} mm2".format(self.AA/mm2_unit))
        print("Core concrete area Ac = {} mm2".format(self.AcAc/mm2_unit))
        print("Unconfined concrete compressive strength fc = {} MPa".format(self.fcfc/MPa_unit))
        print("Young modulus for concrete Ec = {} GPa".format(self.EcEc/GPa_unit))
        print("Diameter of the reinforcing bars D_bars = {} mm and area of one bar Ay = {} mm2 with {} bars".format(self.D_barsD_bars/mm_unit, self.AyAy/mm2_unit, self.nr_barsnr_bars))
        print("Diameter of the hoops D_hoops = {} mm and area of one stirrup As = {} mm2".format(self.D_hoopsD_hoops/mm_unit, self.AsAs/mm2_unit))
        print("Ratio of area of longitudinal reinforcement to area of concrete section rho_bars = {}".format(self.rho_barsrho_bars))
        print("Ratio of area of lateral reinforcement to lateral area of concrete section in x rho_s_x = {} ".format(self.rho_s_xrho_s_x))
        print("Ratio of area of lateral reinforcement to lateral area of concrete section in y rho_s_y = {} ".format(self.rho_s_yrho_s_y))
        print("Moment of inertia of the circular section (strong axis) Iy = {} mm4".format(self.IyIy/mm4_unit))
        print("Moment of inertia of the circular section (weak axis) Iz = {} mm4".format(self.IzIz/mm4_unit))
        print("")


    def ComputeNrBars(self):
        """
        Compute the number of vertical bars in the array bars_position_x (note that this list of lists can have different list sizes).

        @returns int: Number of vertical reinforcing bars.
        """
        nr_bars = 0
        for range in self.bars_position_xbars_position_x:
            nr_bars += np.size(range)-1

        return nr_bars


    def ComputeEc(self):
        """
        Compute Ec using the formula from Mander et Al. 1988.

        @returns float: Young modulus of concrete.
        """

        return 5000.0 * math.sqrt(-self.fcfc/MPa_unit) * MPa_unit


    def ComputeA(self):
        """
        Compute the area for a rectangular section.

```

```
        @returns float: Total area.
        """
        return self.bb * self.dd


    def ComputeAc(self):
        """
        Compute the confined area (area inside the centerline of the hoops, according to Mander et Al. 1988).

        @returns float: Confined area.
        """
        return self.bcbc * self.dcdc


    def ComputeIy(self):
        """
        Compute the moment of inertia of the rectangular section with respect to the strong axis.

        @returns float: Moment of inertia (strong axis)
        """
        return self.bb * self.dd**3 / 12.0


    def ComputeIz(self):
        """
        Compute the moment of inertia of the rectangular section with respect to the weak axis.

        @returns float: Moment of inertia (weak axis)
        """
        return self.dd * self.bb**3 / 12.0


class RCSquareShape(RCRectShape):
    """
    Class that is the children of RCRectShape and cover the specific case of square RC sections.

    @param RCRectShape: Parent class.
    """
    def __init__(self, b, L, e, fc, D_bars, bars_position_x: np.ndarray, bars_ranges_position_y: np.ndarray, fy, Ey, D_hoops, s, fs, Es, name_tag="Not Defined", rho_s_x=-1, rho_s_y=-1, Ec=-1):
        """
        Constructor of the class. It passes the arguments into the parent class to generate the specific case of a aquare RC section.

        @param b (float): Width/depth of the section.
        @param L (float): Effective length of the element associated with this section.
            If the panel zone is present, exclude its dimension.
        @param e (float): Concrete cover.
        @param fc (float): Unconfined concrete compressive strength (cylinder test).
        @param D_bars (float): Diameter of the reinforcing bars.
        @param bars_position_x (np.ndarray):  Distances from border to bar centerline, bar to bar centerlines and
            finally bar centerline to border in the x direction (aligned).
            Starting from the left to right, from the top range to the bottom one.
            The number of bars for each range can vary; in this case, add this argument when defining the array " dtype = object".
        @param bars_ranges_position_y (np.ndarray): Distances from border to range centerlines, range to range centerlines and
            finally range centerline to border in the y direction.
            Starting from the top range to the bottom one.
        @param fy (float): Yield stress for reinforcing bars.
        @param Ey (float): Young modulus for reinforcing bars.
        @param D_hoops (float): Diameter of the hoops.
        @param s (float): Vertical centerline spacing between hoops.
        @param fs (float): Yield stress for the hoops.
        @param Es (float): Young modulus for the hoops
        @param name_tag (str, optional): A nametag for the section. Defaults to "Not Defined".
        @param rho_s_x (float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the x direction.
            Defaults to -1, e.g. computed in __init__()  and ReInit() assuming one range of hoops.
        @param rho_s_y (float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the y direction.
            Defaults to -1, e.g. computed in __init__()  and ReInit() assuming one range of hoops.
        @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in __init__() and ReInit().
        """
        super().__init__(b, b, L, e, fc, D_bars, bars_position_x, bars_ranges_position_y, fy, Ey, D_hoops, s, fs, Es, name_tag, rho_s_x, rho_s_y, Ec)


class RCCircShape(Section):
    """
    Class that stores funcions, geometric and mechanical properties of RC circular shape profile.
    Note that for the validity of the formulas, the hoops needs to be closed (with 135 degress possibly).

```

```
    @param Section: Parent abstract class.
    """
    def __init__(self, b, L, e, fc, D_bars, n_bars: int, fy, Ey, D_hoops, s, fs, Es, name_tag = "Not Defined", rho_s_vol = -1, Ec = -1):
        """
        The conctructor of the class.

        @param b (float): Width of the section.
        @param L (float): Effective length of the element associated with this section.
            If the panel zone is present, exclude its dimension.
        @param e (float): Concrete cover.
        @param fc (float): Unconfined concrete compressive strength (cylinder test).
        @param D_bars (float): Diameter of the vertical reinforcing bars.
        @param n_bars (int): Number of vertical reinforcing bars.
        @param fy (float): Yield stress for reinforcing bars.
        @param Ey (float): Young modulus for reinforcing bars.
        @param D_hoops (float): Diameter of the hoops.
        @param s (float): Vertical centerline spacing between hoops.
        @param fs (float): Yield stress for the hoops.
        @param Es (float): Young modulus for the hoops
        @param name_tag (str, optional): A nametag for the section. Defaults to "Not Defined".
        @param rho_s_vol (float, optional): Ratio of the volume of transverse confining steel to the volume of confined concrete core.
            Defaults to -1, e.g. computed according to Mander et Al. 1988.
        @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in __init__() and ReInit().

        @exception NegativeValue: b needs to be positive.
        @exception NegativeValue: L needs to be positive.
        @exception NegativeValue: e needs to be positive.
        @exception PositiveValue: fc needs to be negative.
        @exception NegativeValue: D_bars needs to be positive.
        @exception NegativeValue: n_bars needs to be a positive integer.
        @exception NegativeValue: fy needs to be positive.
        @exception NegativeValue: Ey needs to be positive.
        @exception NegativeValue: D_hoops needs to be positive.
        @exception NegativeValue: s needs to be positive.
        @exception NegativeValue: fs needs to be positive.
        @exception NegativeValue: Es needs to be positive.
        @exception NegativeValue: Ec needs to be positive if different from -1.
        @exception InconsistentGeometry: e should be smaller than half the depth and the width of the section.
        """
        # Check
        if b < 0: raise NegativeValue()
        if L < 0: raise NegativeValue()
        if e < 0: raise NegativeValue()
        if fc > 0: raise PositiveValue()
        if D_bars < 0: raise NegativeValue()
        if n_bars < 0: raise NegativeValue()
        if fy < 0: raise NegativeValue()
        if Ey < 0: raise NegativeValue()
        if D_hoops < 0: raise NegativeValue()
        if s < 0: raise NegativeValue()
        if fs < 0: raise NegativeValue()
        if Es < 0: raise NegativeValue()
        if Ec != -1 and Ec < 0: raise NegativeValue()
        if e > b/2: raise InconsistentGeometry()

        # Arguments
        self.bb = b
        self.LL = L
        self.ee = e
        self.fcfc = fc
        self.D_barsD_bars = D_bars
        self.n_barsn_bars = n_bars
        self.fyfy = fy
        self.EyEy = Ey
        self.D_hoopsD_hoops = D_hoops
        self.ss = s
        self.fsfs = fs
        self.EsEs = Es
        self.name_tagname_tag = name_tag

        # Initialized the parameters that are dependent from others
        self.ReInitReInit(rho_s_vol, Ec)


    def ReInit(self, rho_s_vol = -1, Ec = -1):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.

        @param rho_s_vol (float, optional): Ratio of the volume of transverse confining steel to the volume of confined concrete core.
            Defaults to -1, e.g. computed according to Mander et Al. 1988.
        @param Ec (float): Young modulus for concrete. Defaults to -1, e.g. computed according to
```

```
        Mander et Al. 1988.
        """
        # Precompute some members
        self.cl_hoopscl_hoops = self.ee + self.D_hoopsD_hoops/2.0 # centerline distance from the border of the extreme confining hoops
        self.cl_barscl_bars = self.ee + self.D_barsD_bars/2.0 + self.D_hoopsD_hoops # centerline distance from the border of the corner bars
        self.bcbc = self.bb - self.cl_hoopscl_hoops*2 # diameter of spiral (hoops) between bar centerline
        self.AsAs = ComputeACircle(self.D_hoopsD_hoops)

        # Arguments
        self.rho_s_volrho_s_vol = self.ComputeRhoVolComputeRhoVol() if rho_s_vol == -1 else rho_s_vol
        self.EcEc = self.ComputeEcComputeEc() if Ec == -1 else Ec

        # Members
        self.AA = ComputeACircle(self.bb)
        self.AcAc = ComputeACircle(self.bcbc)
        self.AyAy = ComputeACircle(self.D_barsD_bars)
        self.rho_barsrho_bars = ComputeRho(self.AyAy, self.n_barsn_bars, self.AA)
        self.II = self.ComputeIComputeI()

        # Data storage for loading/saving
        self.UpdateStoredDataUpdateStoredData()


    def UpdateStoredData(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        self.datadata = [["INFO_TYPE", "RCCircShape"], # Tag for differentiating different data
            ["name_tag", self.name_tagname_tag],
            ["b", self.bb],
            ["bc", self.bcbc],
            ["L", self.LL],
            ["e", self.ee],
            ["A", self.AA],
            ["Ac", self.AcAc],
            ["I", self.II],
            ["fc", self.fcfc],
            ["Ec", self.EcEc],
            ["D_bars", self.D_barsD_bars],
            ["n_bars", self.n_barsn_bars],
            ["Ay", self.AyAy],
            ["rho_bars", self.rho_barsrho_bars],
            ["cl_bars", self.cl_barscl_bars],
            ["fy", self.fyfy],
            ["Ey", self.EyEy],
            ["D_hoops", self.D_hoopsD_hoops],
            ["s", self.ss],
            ["As", self.AsAs],
            ["rho_s_vol", self.rho_s_volrho_s_vol],
            ["cl_hoops", self.cl_hoopscl_hoops],
            ["fs", self.fsfs],
            ["Es", self.EsEs]]

    def ShowInfo(self):
        """
        Implementation of the homonym abstract method.
        See parent class DataManagement for detailed information.
        """
        print("")
        print("Requested info for RC circular section of name tag = {}".format(self.name_tagname_tag))
        print("Width of the section b = {} mm".format(self.bb/mm_unit))
        print("Concrete cover e = {} mm".format(self.ee/mm_unit))
        print("Concrete area A = {} mm2".format(self.AA/mm2_unit))
        print("Core concrete area Ac = {} mm2".format(self.AcAc/mm2_unit))
        print("Unconfined concrete compressive strength fc = {} MPa".format(self.fcfc/MPa_unit))
        print("Young modulus for concrete Ec = {} GPa".format(self.EcEc/GPa_unit))
        print("Diameter of the reinforcing bars D_bars = {} mm and area of one bar Ay = {} mm2 with {} bars".format(self.D_barsD_bars/mm_unit, self.AyAy/mm2_unit, self.n_barsn_bars))
        print("Diameter of the hoops D_hoops = {} mm and area of one stirrup As = {} mm2".format(self.D_hoopsD_hoops/mm_unit, self.AsAs/mm2_unit))
        print("Ratio of area of longitudinal reinforcement to area of concrete section rho_bars = {} ".format(self.rho_barsrho_bars))
        print("Ratio of the volume of transverse confining steel to the volume of confined concrete core rho_s = {} ".format(self.rho_s_volrho_s_vol))
        print("Moment of inertia of the circular section I = {} mm4".format(self.II/mm4_unit))
        print("")


    def ComputeRhoVol(self):
        """
        Compute the ratio of the volume of transverse confining steel to the volume of confined concrete core.
        (according to Mander et Al. 1988).
```

```

        @returns float: Ratio.
        """
        vol_s = self.AsAs*math.pi*self.bcbc
        vol_c = math.pi/4*self.bcbc**2*self.ss

        return vol_s/vol_c


    def ComputeEc(self):
        """
        Compute Ec using the formula from Mander et Al. 1988.

        @returns float: Young modulus of concrete.
        """

        return 5000.0 * math.sqrt(-self.fcfc/MPa_unit) * MPa_unit


    def ComputeI(self):
        """
        Compute the moment of inertia of the circular section.

        @returns float: Moment of inertia.
        """
        return self.bb**4*math.pi/64


def ComputeACircle(D):
    """
    Function that computes the area of one circle (reinforcing bar or hoop).

    @param D (float): Diameter of the circle (reinforcing bar of hoop).

    @returns float: Area the circle (for reinforcing bars or hoops).
    """
    return D**2/4.0*math.pi


def ComputeRho(A, nr, A_tot):
    """
    Compute the ratio of area of a reinforcement to area of a section.

    @param A (float): Area of reinforcement.
    @param nr (float): Number of reinforcement (allow float for computing ratio with different area;
        just convert the other areas to one and compute the equivalent n).
    @param A_tot (float): Area of the concrete.

    @returns float: Ratio.
    """
    return nr * A / A_tot
```

8.24 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Units.py File Reference

Namespaces

- namespace Units

  Module with the units conversion and the definition of the units used as default (m, N, s).

Variables

- float cm2_unit = cm_unit∗cm_unit
- float cm3_unit = cm_unit∗cm_unit∗cm_unit
- float cm4_unit = cm3_unit∗cm_unit
- float cm_unit = m_unit∗1e-2
- float dm2_unit = dm_unit∗dm_unit
- float dm3_unit = dm_unit∗dm_unit∗dm_unit
- float dm4_unit = dm3_unit∗dm_unit
- float dm_unit = m_unit∗1e-1

- string force_unit = "N"
- float ft2_unit = ft_unit*ft_unit
- float ft3_unit = ft_unit*ft_unit*ft_unit
- float ft4_unit = ft3_unit*ft_unit
- float ft_unit = m_unit*0.3048
- float GN_unit = N_unit*1e9
- float GPa_unit = Pa_unit*1e9
- float hours_unit = min_unit*60
- float inch2_unit = inch_unit*inch_unit
- float inch3_unit = inch_unit*inch_unit*inch_unit
- float inch4_unit = inch3_unit*inch_unit
- float inch_unit = m_unit*0.0254
- float kg_unit = N_unit*s_unit**2/m_unit
- float kip_unit = N_unit*4448.2216
- float km_unit = m_unit*1e3
- float kN_unit = N_unit*1e3
- float kNm_unit = kN_unit*m_unit
- float kNmm_unit = kN_unit*mm_unit
- float kPa_unit = Pa_unit*1e3
- float ksi_unit = psi_unit*1000
- string length_unit = "m"
- float m2_unit = m_unit*m_unit
- float m3_unit = m_unit*m_unit*m_unit
- float m4_unit = m3_unit*m_unit
- float m_unit = 1.0
- float mile_unit = m_unit*1609.34
- float min_unit = s_unit*60
- float mm2_unit = mm_unit*mm_unit
- float mm3_unit = mm_unit*mm_unit*mm_unit
- float mm4_unit = mm3_unit*mm_unit
- float mm_unit = m_unit*1e-3
- float MN_unit = N_unit*1e6
- float MNm_unit = MN_unit*m_unit
- float MNmm_unit = MN_unit*mm_unit
- float MPa_unit = Pa_unit*1e6
- float N_unit = 1.0
- float Nm_unit = N_unit*m_unit
- float Nmm_unit = N_unit*mm_unit
- float Pa_unit = N_unit/m2_unit
- float pound_unit = kg_unit*0.45359237
- float psi_unit = Pa_unit*6894.76
- float s_unit = 1.0
- float t_unit = kg_unit*1e3
- string time_unit = "s"

8.25 Units.py

Go to the documentation of this file.

```
"""Module with the units conversion and the definition of the units used as default (m, N, s). \n
Note that the decision of which unit for each measure (distance, force, mass, time) is equal to 1 is not arbitrary:
for example the natural frequency is computed behind the scene by the OpenSeesPy framework, thus the stiffness of the structure divided by the mass should result in a unit of 1 (thus seconds). \n
Furthermore, there are constants like the gravitational one g that is dependent on this decision. If the units are used in a consistent way (using this library), these issues can be avoided. \n
```

```
Carmine Schipani, 2021
"""


# Fundamental
m_unit = 1.0
length_unit = "m" # It’s the length unit associated with 1 (fundamental)
N_unit = 1.0
force_unit = "N" # It’s the force unit associated with 1 (fundamental)
s_unit = 1.0
time_unit = "s" # It’s the time unit associated with 1 (fundamental)

# Distance
mm_unit = m_unit*1e-3
cm_unit = m_unit*1e-2
dm_unit = m_unit*1e-1
km_unit = m_unit*1e3
inch_unit = m_unit*0.0254
ft_unit = m_unit*0.3048
mile_unit = m_unit*1609.34

# Area
mm2_unit = mm_unit*mm_unit
cm2_unit = cm_unit*cm_unit
dm2_unit = dm_unit*dm_unit
m2_unit = m_unit*m_unit
inch2_unit = inch_unit*inch_unit
ft2_unit = ft_unit*ft_unit

# Volume
mm3_unit = mm_unit*mm_unit*mm_unit
cm3_unit = cm_unit*cm_unit*cm_unit
dm3_unit = dm_unit*dm_unit*dm_unit
m3_unit = m_unit*m_unit*m_unit
inch3_unit = inch_unit*inch_unit*inch_unit
ft3_unit = ft_unit*ft_unit*ft_unit

# Moment of inertia
mm4_unit = mm3_unit*mm_unit
cm4_unit = cm3_unit*cm_unit
dm4_unit = dm3_unit*dm_unit
m4_unit = m3_unit*m_unit
inch4_unit = inch3_unit*inch_unit
ft4_unit = ft3_unit*ft_unit

# Force
kN_unit = N_unit*1e3
MN_unit = N_unit*1e6
GN_unit = N_unit*1e9
kip_unit = N_unit*4448.2216

# Moment (and rotational stiffnes (moment-rotation))
Nm_unit = N_unit*m_unit
kNm_unit = kN_unit*m_unit
MNm_unit = MN_unit*m_unit
Nmm_unit = N_unit*mm_unit
kNmm_unit = kN_unit*mm_unit
MNmm_unit = MN_unit*mm_unit

# Mass
kg_unit = N_unit*s_unit**2/m_unit
t_unit = kg_unit*1e3
pound_unit = kg_unit*0.45359237

# Pressure/Stress
Pa_unit = N_unit/m2_unit
kPa_unit = Pa_unit*1e3
MPa_unit = Pa_unit*1e6
GPa_unit = Pa_unit*1e9
psi_unit = Pa_unit*6894.76
ksi_unit = psi_unit*1000

# Time
min_unit = s_unit*60
hours_unit = min_unit*60

```

Index