opsvis

Release 0.99.1

Seweryn Kokot

Jul 08, 2023

CONTENTS:

CHAPTER

ONE

PLOT_MODEL

opsvis.**plot_model**(*node_labels=1*, *element_labels=1*, *offset_nd_label=False*, *axis_off=0*, *az_el=(-60.0, 30.0)*, *fig_wi_he=False*, *fig_lbrt=False*, *local_axes=True*, *nodes_only=False*, *fmt_model={'color': 'blue', 'linestyle': 'solid', 'linewidth': 1.2, 'marker': '.', 'markersize': 6}*, *fmt_model_nodes_only={'color': 'blue', 'linestyle': 'solid', 'linewidth': 1.2, 'marker': '.', 'markersize': 6}*, *node_supports=True*, *gauss_points=True*, *fmt_gauss_points={'color': 'firebrick', 'linestyle': 'None', 'linewidth': 2.0, 'marker': 'X', 'markersize': 5}*, *fmt_model_truss={'color': 'green', 'linestyle': 'solid', 'linewidth': 1.2, 'marker': 'o', 'markerfacecolor': 'white', 'markersize': 6}*, *truss_node_offset=0.96*, *ax=False*)

Plot defined model of the structure.

Parameters

- **node_labels** (*int*) – 1 - plot node labels, 0 - do not plot them; (default: 1)
- **element_labels** (*int*) – 1 - plot element labels, 0 - do not plot them; (default: 1)
- **offset_nd_label** (*bool*) – False - do not offset node labels from the actual node location. This option can enhance visibility.
- **axis_off** (*int*) – 0 - turn off axes, 1 - display axes; (default: 0)
- **az_el** (*tuple*) – contains azimuth and elevation for 3d plots. For 2d plots this parameter is neglected.
- **fig_wi_he** (*tuple*) – contains width and height of the figure
- **fig_lbrt** (*tuple*) – a tuple contating left, bottom, right and top offsets
- **local_axes** (*bool*) – True - show cross section local axes or False. The green, red and blue arrows denote the element axis direction, the z-local axis and the y-local axis.
- **nodes_only** (*bool*) – True - show the nodes only, although the elements are defined. Default: False.
- **fmt_model** (*dict*) – A dictionary containing formatting the line and markers of the model elements. The formatting options can be: linewidth, color, marker, markersize. See matplotlib.plot documentation for more details, if necessary.
- **node_supports** (*bool*) – True - show the supports. Default: True.
- **gauss_points** (*bool*) – True - show the integration (Gauss) points. Default: True.
- **fmt_gauss_points** (*dict*) – A dictionary containing formatting the marker of the gauss point.
- **truss_node_offset** (*float*) – If non-zero, the nodes are offset to show pin markers. The number should preferably be between 0.90-0.97. Zero (0) or False means no offset. Default: 0.96.

- **ax** – axis object.

Usage:

`plot_model()` - plot model with node and element labels.

`plot_model(node_labels=0, element_labels=0)` - plot model without node element labels

`plot_model(fig_wi_he=(20., 14.))` - plot model in a window 20 cm long, and 14 cm high.

`plot_model(nodes_only=True)` - plot only the nodes even thought the elements are defined.

`plot_model(node_supports=False)` - plot the model without the supports.

CHAPTER

TWO

PLOT_DEFO

opsvis.**plot_defo**(*sfac=False*, *nep=17*, *unDefoFlag=1*, *fmt_defo={'color': 'blue', 'linestyle': 'solid', 'linewidth': 1.2, 'marker': '', 'markersize': 1}*, *fmt_undefo={'color': 'green', 'linestyle': (0, (1, 5)), 'linewidth': 1.2, 'marker': '', 'markersize': 1}*, *fmt_defo_faces={'alpha': 0.5, 'edgecolors': 'k', 'linewidths': 1}*, *fmt_undefo_faces={'alpha': 0.5, 'edgecolors': 'g', 'facecolors': 'w', 'linestyles': 'dotted', 'linewidths': 1}*, *interpFlag=1*, *endDispFlag=0*, *fmt_nodes={'color': 'red', 'linestyle': 'None', 'linewidth': 1.2, 'marker': 's', 'markersize': 6}*, *Eo=0*, *az_el=(-60.0, 30.0)*, *fig_wi_he=False*, *fig_lbrt=False*, *node_supports=True*, *ax=False*)

Plot deformed shape of the structure.

Parameters

- **sfac** (*float*) – scale factor to increase/decrease displacements obtained from FE analysis. If not specified (False), sfac is automatically calculated based on the maximum overall displacement and this maximum displacement is plotted as 20 percent (hardcoded) of the maximum model dimension.
- **interpFlag** (*int*) – 1 - use interpolated deformation using shape function, 0 - do not use interpolation, just show displaced element nodes (default is 1)
- **nep** (*int*) – number of evaluation points for shape function interpolation (default: 17)
- **node_supports** (*bool*) – True - show the supports. Default: True.

Returns

the automatically calculated scale factor can be
returned.

Return type
sfac (float)

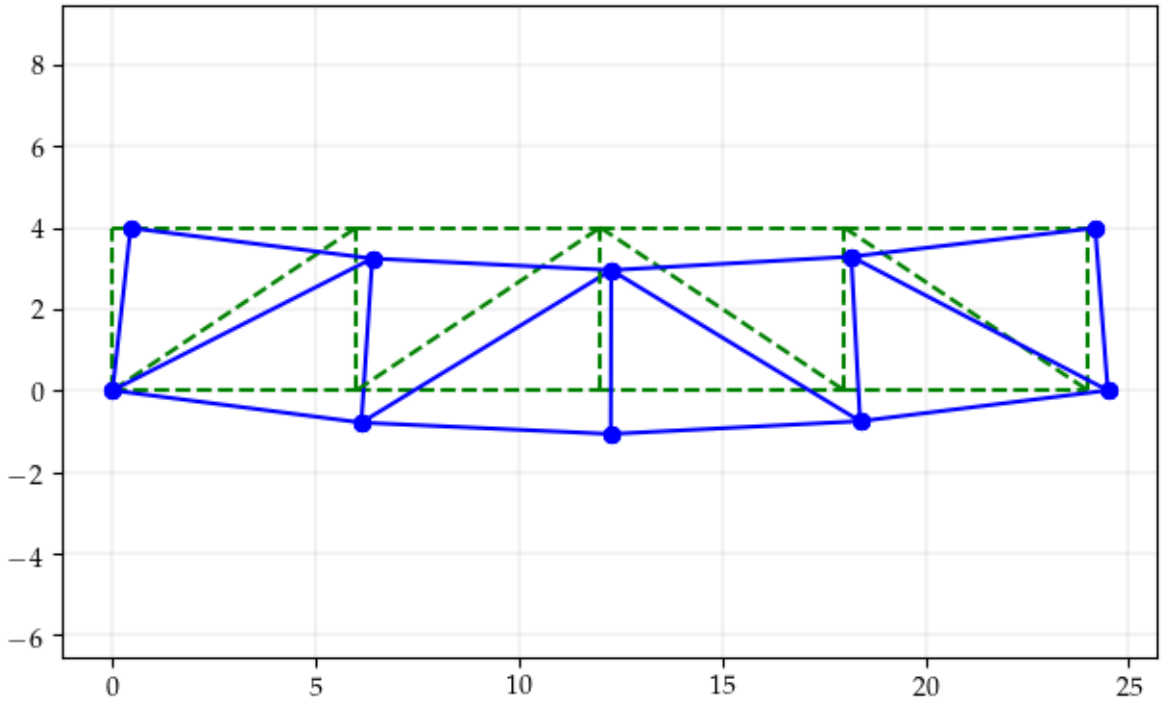
Usage:

`sfac = plot_defo()` - plot deformed shape with default parameters and automatically calcutated scale factor.

`plot_defo(interpFlag=0)` - plot simplified deformation by displacing the nodes connected with straight lines (shape function interpolation)

`plot_defo(sfac=1.5)` - plot with specified scale factor

`plot_defo(unDefoFlag=0, endDispFlag=0)` - plot without showing undeformed (original) mesh and without showing markers at the element ends.


8
6
4
2
0
−2
−4
−6
0
5
10
15
20
25

CHAPTER

THREE

PLOT_LOADS_2D

opsvis.**plot_loads_2d**(*nep=17*, *sfac=False*, *fig_wi_he=False*, *fig_lbrt=False*, *fmt_model_loads={'color': 'black', 'linestyle': 'solid', 'linewidth': 1.2, 'marker': '', 'markersize': 1}*, *node_supports=True*, *truss_node_offset=0*, *ax=False*)

Display the nodal and element loads applied to the 2d models.

Parameters

- **nep** (*int*) – number of arrows when displacing element distributed loads (default: 17)
- **node_supports** (*bool*) – True - show the node support conditions. Default: False.

CHAPTER

FOUR

PLOT_MODE_SHAPE

opsvis.**plot_mode_shape**(*modeNo*, *sfac=False*, *nep=17*, *unDefoFlag=1*, *fmt_defo={'color': 'blue', 'linestyle': 'solid', 'linewidth': 1.2, 'marker': '', 'markersize': 1}*, *fmt_undefo={'color': 'green', 'linestyle': (0, (1, 5)), 'linewidth': 1.2, 'marker': '', 'markersize': 1}*, *fmt_defo_faces={'alpha': 0.5, 'edgecolors': 'k', 'linewidths': 1}*, *fmt_undefo_faces={'alpha': 0.5, 'edgecolors': 'g', 'facecolors': 'w', 'linestyles': 'dotted', 'linewidths': 1}*, *interpFlag=1*, *endDispFlag=1*, *fmt_nodes={'color': 'red', 'linestyle': 'None', 'linewidth': 1.2, 'marker': 's', 'markersize': 6}*, *Eo=0*, *az_el=(-60.0, 30.0)*, *fig_wi_he=False*, *fig_lbrt=False*, *node_supports=True*, *ax=False*)

Plot mode shape of the structure obtained from eigenvalue analysis.

Parameters

- **modeNo** (*int*) – indicates which mode shape to plot
- **sfac** (*float*) – scale factor to increase/decrease displacements obtained from FE analysis. If not specified (False), sfac is automatically calculated based on the maximum overall displacement and this maximum displacement is plotted as 20 percent (hardcoded) of the maximum model dimension.
- **interpFlag** (*int*) – 1 - use interpolated deformation using shape function, 0 - do not use interpolation, just show displaced element nodes (default is 1)
- **nep** (*int*) – number of evaluation points for shape function interpolation (default: 17)

Usage:

`plot_mode_shape(1)` - plot the first mode shape with default parameters and automatically calcutated scale factor.

`plot_mode_shape(2, interpFlag=0)` - plot the 2nd mode shape by displacing the nodes connected with straight lines (shape function interpolation)

`plot_mode_shape(3, sfac=1.5)` - plot the 3rd mode shape with specified scale factor

`plot_mode_shape(4, unDefoFlag=0, endDispFlag=0)` - plot the 4th mode shape without showing undeformed (original) mesh and without showing markers at the element ends.

Examples:

Notes:

See also:

plot_defo()

CHAPTER

FIVE

SECTION_FORCE_DIAGRAM_2D

opsvis.**section_force_diagram_2d**(*sf_type*, *sfac=1.0*, *nep=17*, *fmt_secforce1={'color': 'blue', 'dash_capstyle': 'butt', 'dash_joinstyle': 'round', 'linestyle': 'solid', 'linewidth': 2.0, 'marker': '', 'markersize': 1, 'solid_capstyle': 'round', 'solid_joinstyle': 'round'}*, *fmt_secforce2={'color': 'blue', 'dash_capstyle': 'butt', 'dash_joinstyle': 'round', 'linestyle': 'solid', 'linewidth': 1.0, 'marker': '', 'markersize': 1, 'solid_capstyle': 'round', 'solid_joinstyle': 'round'}*, *fig_wi_he=False*, *fig_lbrt=False*, *ref_vert_lines=True*, *end_max_values=True*, *node_supports=True*, *ax=False*, *alt_model_plot=1*)

Display section forces diagram for 2d beam column model.

This function plots a section forces diagram for 2d beam column elements with or without element loads. For now only '-beamUniform' constant transverse or axial element loads are supported.

Parameters

- **sf_type** (*str*) – type of section force: 'N' - normal force, 'V' - shear force, 'M' - bending moments.
- **sfac** (*float*) – scale factor by wich the values of section forces are multiplied.
- **nep** (*int*) – number of evaluation points including both end nodes (default: 17)
- **fmt_secforce1** (*dict*) – line format dictionary for section force distribution curve.
- **fmt_secforce2** (*dict*) – line format dictionary for auxiliary reference lines.
- **fig_wi_he** (*tuple*) – contains width and height of the figure
- **fig_lbrt** (*tuple*) – a tuple contating left, bottom, right and top offsets
- **ref_vert_lines** (*bool*) – True means plot the vertical reference lines on the section force diagrams.
- **end_max_values** (*bool*) – True means show the values at element ends and extreme (max, min) value between the ends.
- **node_supports** (*bool*) – True - show the supports. Default: True.
- **ax** – the axes object.

Returns

the minimum overall value of the section force.

maxVal (float): the maximum overall value of the section force.

ax: the axes object.

alt_model_plot (int): 1 - for using the plot_model command, 2 - for using
simplified model plotting. Other integer - for no model plotting. In this case the model can be plotted outside this command using the axes (ax) object. Default is 1.

Return type
minVal (float)

Usage:
See example: demo_portal_frame.py

See example *2d Portal Frame*

CHAPTER

SIX

SECTION_FORCE_DIAGRAM_3D

opsvis.**section_force_diagram_3d**(*sf_type*, *sfac=1.0*, *nep=17*, *fmt_secforce1={'color': 'blue', 'dash_capstyle': 'butt', 'dash_joinstyle': 'round', 'linestyle': 'solid', 'linewidth': 2.0, 'marker': '', 'markersize': 1, 'solid_capstyle': 'round', 'solid_joinstyle': 'round'}*, *fmt_secforce2={'color': 'blue', 'dash_capstyle': 'butt', 'dash_joinstyle': 'round', 'linestyle': 'solid', 'linewidth': 1.0, 'marker': '', 'markersize': 1, 'solid_capstyle': 'round', 'solid_joinstyle': 'round'}*, *ref_vert_lines=True*, *end_max_values=True*, *dir_plt=0*, *node_supports=True*, *ax=False*, *alt_model_plot=1*)

Display section forces diagram of a 3d beam column model.

This function plots section forces diagrams for 3d beam column elements with or without element loads. For now only '-beamUniform' constant transverse or axial element loads are supported.

Parameters

- **sf_type** (*str*) – type of section force: 'N' - normal force, 'Vy' or 'Vz' - shear force, 'My' or 'Mz' - bending moments, 'T' - torsional moment.
- **sfac** (*float*) – scale factor by wich the values of section forces are multiplied.
- **nep** (*int*) – number of evaluation points including both end nodes (default: 17)
- **fmt_secforce1** (*dict*) – line format dictionary for section force distribution curve.
- **fmt_secforce2** (*dict*) – line format dictionary for auxiliary reference lines.
- **end_max_values** (*bool*) – True means show the values at element ends and extreme (max, min) value between the ends.
- **{0** (*dir_plt*) – direction in which to plot the load effects: 0 (default) - as defined in the code for each load effect type 1 - in the y-axis (default for N, Vy, T, Mz) 2 - in the z-axis (default for Vz, My)
- **1** – direction in which to plot the load effects: 0 (default) - as defined in the code for each load effect type 1 - in the y-axis (default for N, Vy, T, Mz) 2 - in the z-axis (default for Vz, My)
- **2}** – direction in which to plot the load effects: 0 (default) - as defined in the code for each load effect type 1 - in the y-axis (default for N, Vy, T, Mz) 2 - in the z-axis (default for Vz, My)
- **ax** – Optional axis to plot to.
- **alt_model_plot** (*int*) – 1 - for using the plot_model command, 2 - for using simplified model plotting. Other integer - for no model plotting. In this case the model can be plotted outside this command using the axes (ax) object. Default is 1.

Returns

the minimum overall value of the section force.

maxVal (float): the maximum overall value of the section force.

ax: the axes object.

Return type
minVal (float)

Usage:
See example: demo_cantilever_3el_3d.py

Todo:

Add support for other element loads available in OpenSees: partial (trapezoidal) uniform element load, and 'beamPoint' element load.

See example *Statics of a 3d 3-element cantilever beam*

CHAPTER

SEVEN

PLOT_STRESS_2D

opsvis.**plot_stress_2d**(*nds_val*, *mesh_outline=1*, *cmap='turbo'*, *levels=50*)

Plot stress distribution of a 2d elements of a 2d model.

Parameters

- **nds_val** (*ndarray*) – the values of a stress component, which can be extracted from sig_out array (see sig_out_per_node function)
- **mesh_outline** (*int*) – 1 - mesh is plotted, 0 - no mesh plotted.
- **cmap** (*str*) – Matplotlib color map (default is 'turbo')

Usage:

See demo_quads_4x4.py example.

See example *Plot stress distribution of a plane stress quad model*

CHAPTER

EIGHT

PLOT_EXTRUDED_MODEL_RECT_SECTION_3D

opsvis.**plot_extruded_shapes_3d**(*ele_shapes*, *az_el=(-60.0, 30.0)*, *fig_wi_he=False*, *fig_lbrt=False*, *ax=False*)

Plot an extruded 3d model based on cross-section dimenions.

Three arrows present local section axes: green - local x-axis, red - local z-axis, blue - local y-axis.

Parameters

- **ele_shapes** (*dict*) – keys are ele_tags and values are lists of: shape_type (str): 'rect' - rectangular shape, 'I' - double T shape and shape_args (list): list of floats, which necessary section dimensions. For 'rect' the list is [b d]; width and depth, for 'I' shape - [bf d tw tf]; flange width, section depth, web and flange thicknesses Example: ele_shapes = {1: ['rect', [b, d]], 2: ['I', [bf, d, tw, tf]]}
- **az_el** (*tuple*) – azimuth and elevation
- **fig_wi_he** – figure width and height in centimeters
- **fig_lbrt** – figure left, bottom, right, top boundaries

Usage:

```
ele_shapes = {1: ['circ', [b]],
              2: ['rect', [b, h]],
              3: ['I', [b, h, b/10., h/6.]]}
opsv.plot_extruded_shapes_3d(ele_shapes)
```

Notes:

- For now only rectangular, circular and double T sections are supported.
- This function can be a source of inconsistency because OpenSees lacks functions to return section dimensions. A workaround is to have own Python helper functions to reuse data specified once

See example *Statics of a 3d 3-element cantilever beam*

CHAPTER

NINE

ANIM_DEFO

opsvis.**anim_defo**(*Eds*, *timeV*, *sfac*, *nep=17*, *unDefoFlag=1*, *fmt_defo={'color': 'blue', 'linestyle': 'solid', 'linewidth': 1.2, 'marker': '', 'markersize': 1}*, *fmt_undefo={'color': 'green', 'linestyle': (0, (1, 5)), 'linewidth': 1.2, 'marker': '', 'markersize': 1}*, *interpFlag=1*, *endDispFlag=1*, *az_el=(-60.0, 30.0)*, *fig_wi_he=False*, *fig_lbrt=False*, *xlim=[0, 1]*, *ylim=[0, 1]*, *ax=False*)

Make animation of the deformed shape computed by transient analysis

Parameters

- **Eds** (*ndarray*) – A 3d array (n_steps x n_eles x n_dof_per_element) containing the collected displacements per element for all time steps.
- **timeV** (*1darray*) – vector of discretized time values
- **sfac** (*float*) – scale factor
- **nep** (*integer*) – number of evaluation points inside the element and including both element ends
- **unDefoFlag** (*integer*) – 1 - plot the undeformed model (mesh), 0 - do not plot the mesh
- **interpFlag** (*integer*) – 1 - interpolate deformation inside element, 0 - no interpolation
- **endDispFlag** (*integer*) – 1 - plot marks at element ends, 0 - no marks
- **fmt_defo** (*dict*) – format line string for interpolated (continuous) deformated shape. The format contains information on line color, style and marks as in the standard matplotlib plot function.
- **az_el** (*tuple*) – a tuple containing the azimuth and elevation
- **fig_lbrt** (*tuple*) – a tuple contating left, bottom, right and top offsets
- **fig_wi_he** (*tuple*) – contains width and height of the figure

Examples:

Notes:

See also:

CHAPTER

TEN

ANIM_MODE

opsvis.**anim_mode**(*modeNo*, *sfac=False*, *nep=17*, *unDefoFlag=1*, *fmt_defo={'color': 'blue', 'linestyle': 'solid', 'linewidth': 1.2, 'marker': '', 'markersize': 1}*, *fmt_undefo={'color': 'green', 'linestyle': (0, (1, 5)), 'linewidth': 1.2, 'marker': '', 'markersize': 1}*, *interpFlag=1*, *endDispFlag=1*, *Eo=0*, *az_el=(-60.0, 30.0)*, *fig_wi_he=False*, *fig_lbrt=False*, *xlim=[0, 1]*, *ylim=[0, 1]*, *ax=False*)

Make animation of a mode shape obtained from eigenvalue solution.

Parameters

- **modeNo** (*int*) – indicates which mode shape to animate.
- **sfac** (*float*) – scale factor
- **nep** (*integer*) – number of evaluation points inside the element and including both element ends
- **unDefoFlag** (*integer*) – 1 - plot the undeformed model (mesh), 0 - do not plot the mesh
- **interpFlag** (*integer*) – 1 - interpolate deformation inside element, 0 - no interpolation
- **endDispFlag** (*integer*) – 1 - plot marks at element ends, 0 - no marks
- **fmt_defo** (*dict*) – format line string for interpolated (continuous) deformated shape. The format contains information on line color, style and marks as in the standard matplotlib plot function.
- **az_el** (*tuple*) – a tuple containing the azimuth and elevation
- **fig_lbrt** (*tuple*) – a tuple contating left, bottom, right and top offsets
- **fig_wi_he** (*tuple*) – contains width and height of the figure

Returns
animation object

Return type
anim

Notes:

See also:

anim_mode()

CHAPTER

ELEVEN

PLOT_FIBER_SECTION

opsvis.**plot_fiber_section**(*fib_sec_list*, *fillflag=1*, *matcolor=['y', 'b', 'r', 'g', 'm', 'k']*)

Plot fiber cross-section.

Parameters

- **fib_sec_list** (*list*) – list of lists in the format similar to the parameters for the section, layer, patch, fiber OpenSees commands
- **fillflag** (*int*) – 1 - filled fibers with color specified in matcolor list, 0 - no color, only the outline of fibers
- **matcolor** (*list*) – sequence of colors for various material tags assigned to fibers

Examples

```
fib_sec_1 = [['section', 'Fiber', 1, '-GJ', 1.0e6],
             ['patch', 'quad', 1, 4, 1,  0.032, 0.317, -0.311, 0.067, -0.266, 0.005,
↪ 0.077, 0.254],  # noqa: E501
             ['patch', 'quad', 1, 1, 4,  -0.075, 0.144, -0.114, 0.116, 0.075, -0.
↪144, 0.114, -0.116],  # noqa: E501
             ['patch', 'quad', 1, 4, 1,  0.266, -0.005,  -0.077, -0.254,  -0.032, -
↪0.317,  0.311, -0.067]  # noqa: E501
             ]
opsv.fib_sec_list_to_cmds(fib_sec_1)
matcolor = ['r', 'lightgrey', 'gold', 'w', 'w', 'w']
opsv.plot_fiber_section(fib_sec_1, matcolor=matcolor)
plt.axis('equal')
# plt.savefig('fibsec_rc.png')
plt.show()
```

Notes

fib_sec_list can be reused by means of a python helper function

```
opsvis.fib_sec_list_to_cmds(fib_sec_list_1)
```

See also:

`opsvis.fib_sec_list_to_cmds()`

See example *Plot steel and reinforced concrete fiber sections*

CHAPTER

TWELVE

FIB_SEC_LIST_TO_CMDS

opsvis.**fib_sec_list_to_cmds**(*fib_sec_list*)

Reuses fib_sec_list to define fiber section in OpenSees.

At present it is not possible to extract fiber section data from the OpenSees domain, this function is a workaround. The idea is to prepare data similar to the one the regular OpenSees commands (`section('Fiber', ...)`, `fiber()`, `patch()` and/or `layer()`) require.

Parameters

- **fib_sec_list** (*list*) – is a list of fiber section data. First sub-list
- **stiffness** (*also defines the torsional*) –

Warning:

If you use this function, do not issue the regular OpenSees: section, Fiber, Patch or Layer commands.

See also:

`opsvis.plot_fiber_section()`

See example *Plot steel and reinforced concrete fiber sections*

CHAPTER

THIRTEEN

SIG_OUT_PER_NODE

opsvis.**sig_out_per_node**(*how_many='all'*)

Return a 2d numpy array of stress components per OpenSees node.

Three first stress components (sxx, syy, sxy) are calculated and extracted from OpenSees, while the rest svm (Huber-Mieses-Hencky), two principal stresses (s1, s2) and directional angle are calculated as postprocessed quantities.

Parameters

how_many (*str*) – supported options are: 'all' - all components, 'sxx', 'syy', 'sxy', 'svm' (or 'vmis'), 's1', 's2', 'angle'.

Returns

a 2d array of stress components per node with
the following components: sxx, syy, sxy, svm, s1, s2, angle. Size (n_nodes x 7).

Return type

sig_out (ndarray)

Examples

sig_out = opsv.sig_out_per_node()

Notes

s1, s2: principal stresses angle: angle of the principal stress s1

See example *Plot stress distribution of a plane stress quad model*

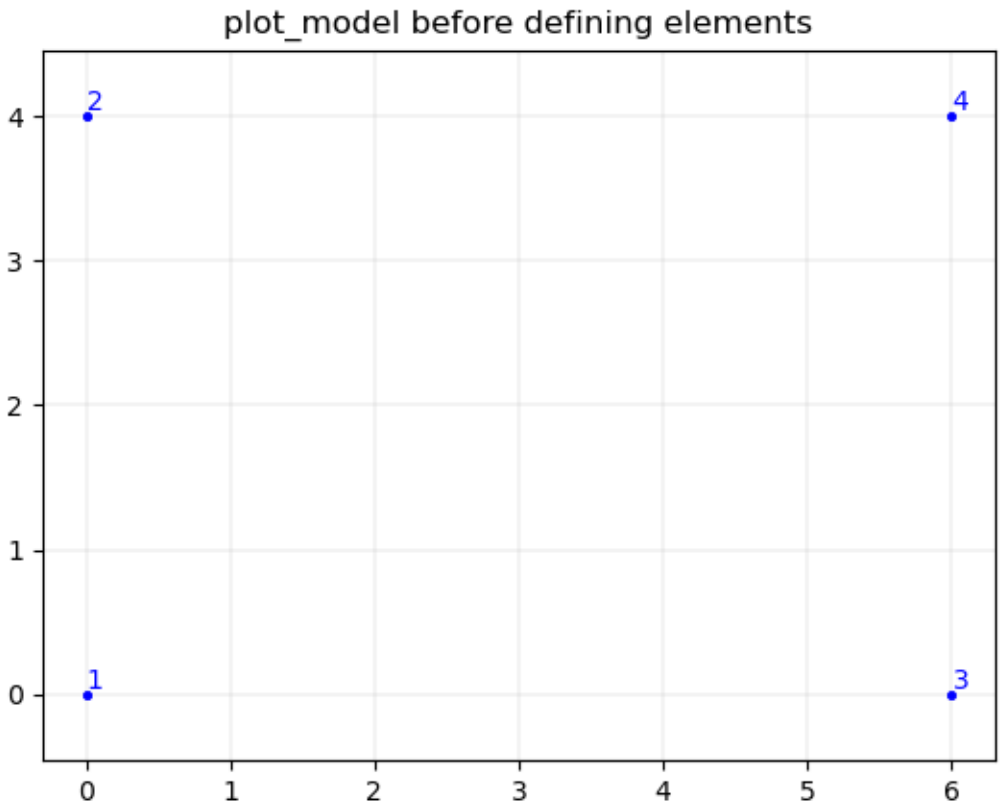
CHAPTER

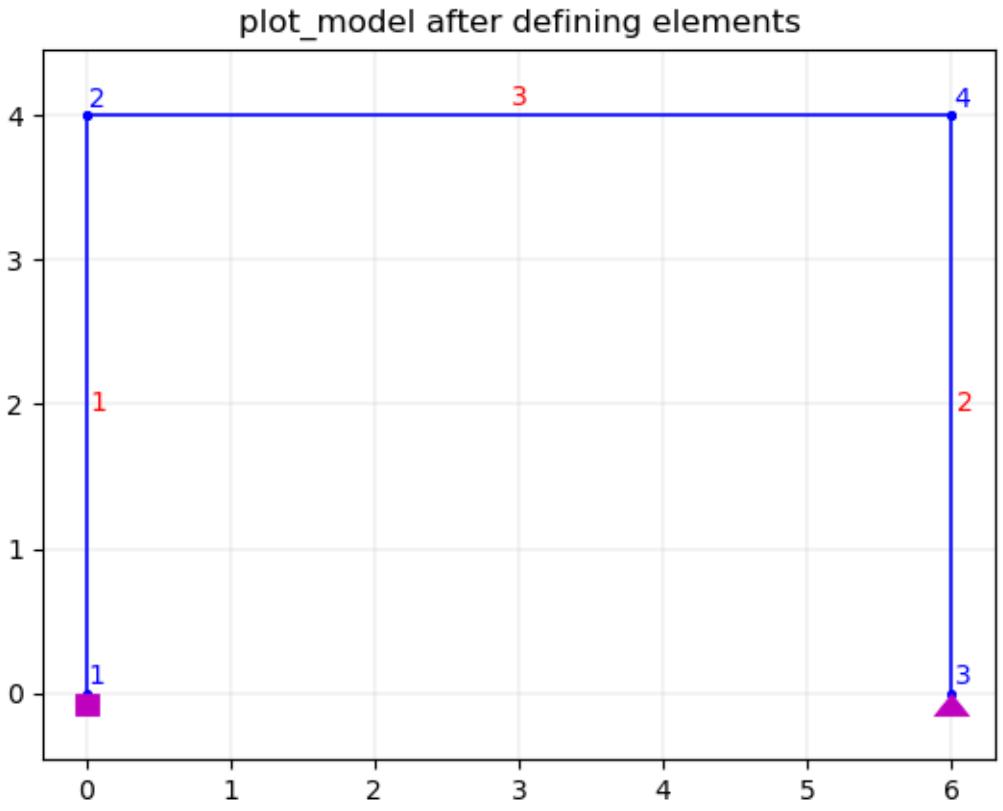
FOURTEEN

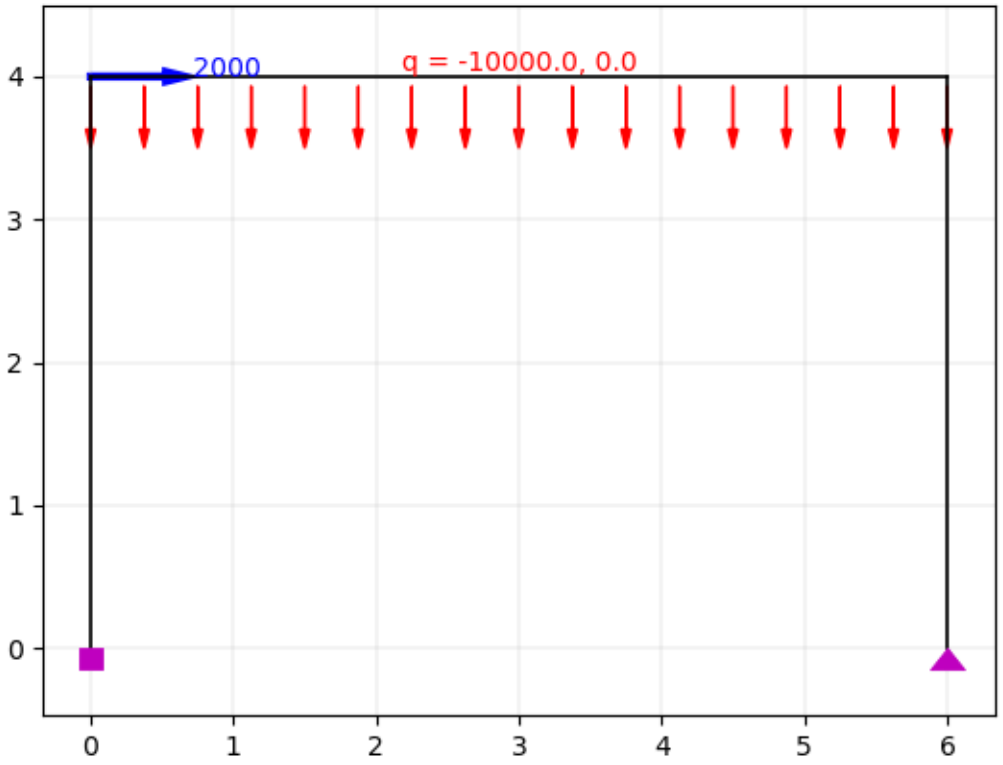
EXAMPLES

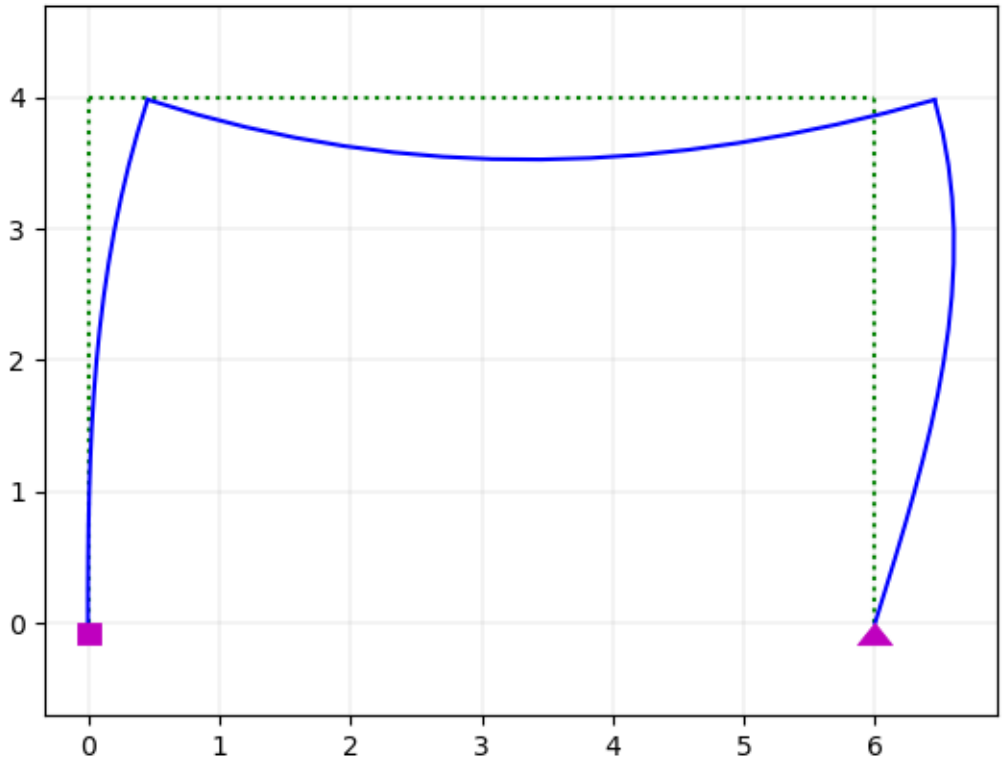
14.1 2d Portal Frame

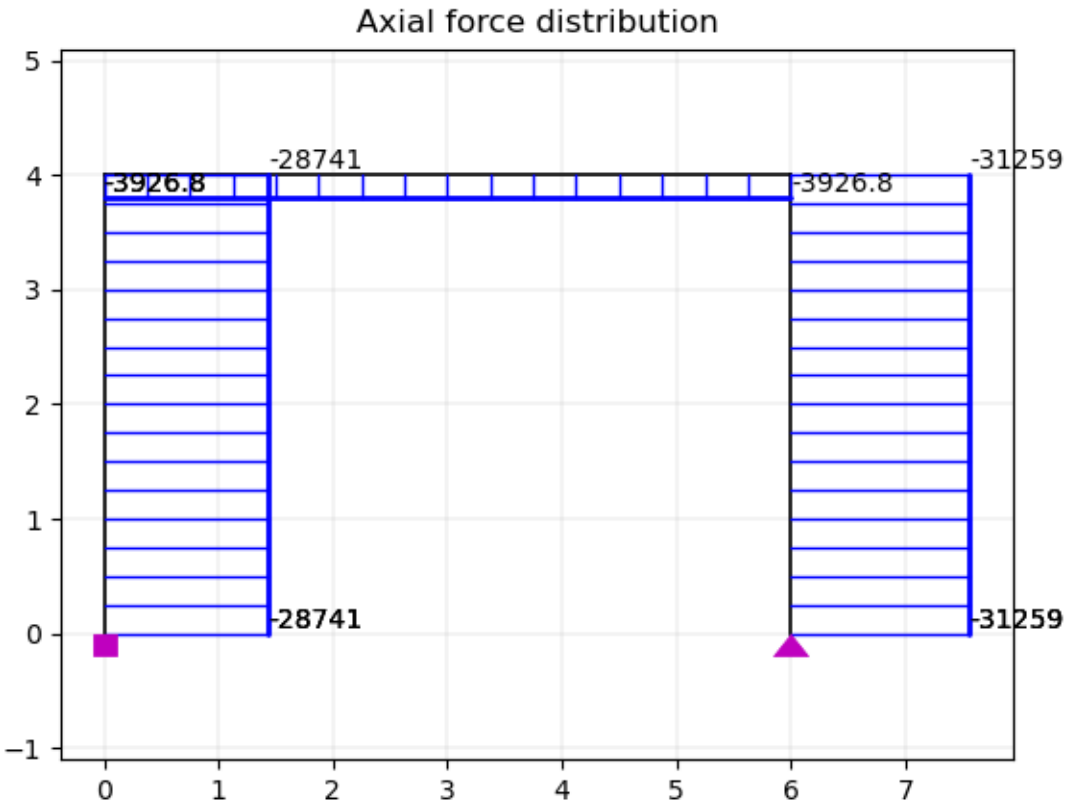
Example `.py` file can be downloaded `here`:

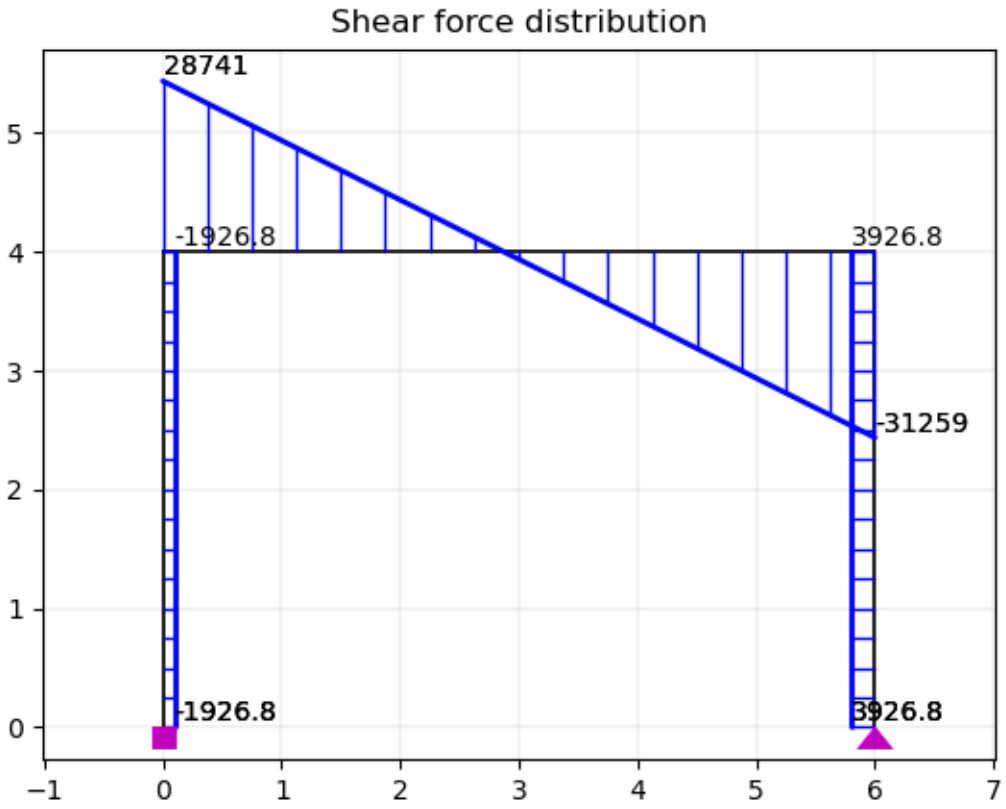


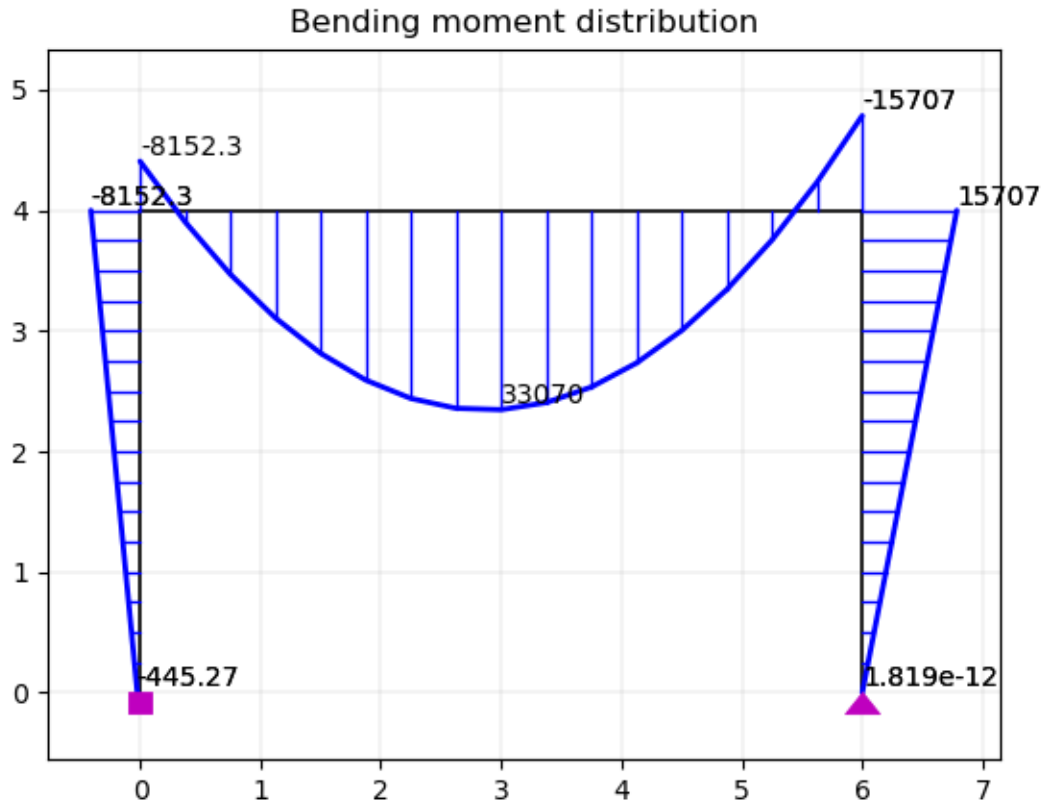

plot_model after defining elements
2
3
4
1
2
1
3
0
1
2
3
4
5
6


2000
q = -10000.0, 0.0
4
3
2
1
0
0
1
2
3
4
5
6


4
3
2
1
0
0
1
2
3
4
5
6


Axial force distribution
-28741
-31259
-3926.8
-3926.8
-28741
-31259
5
4
3
2
1
0
−1
0
1
2
3
4
5
6
7


Shear force distribution
28741
-1926.8
3926.8
-31259
-1926.8
3926.8
5
4
3
2
1
0
−1
0
1
2
3
4
5
6
7




```
import openseespy.opensees as ops
import opsvis as opsv

import matplotlib.pyplot as plt

ops.wipe()
ops.model('basic', '-ndm', 2, '-ndf', 3)

colL, girL = 4., 6.

Acol, Agir = 2.e-3, 6.e-3
IzCol, IzGir = 1.6e-5, 5.4e-5

E = 200.e9

Ep = {1: [E, Acol, IzCol],
      2: [E, Acol, IzCol],
      3: [E, Agir, IzGir]}

ops.node(1, 0., 0.)
ops.node(2, 0., colL)
ops.node(3, girL, 0.)
ops.node(4, girL, colL)
```

(continues on next page)

(continued from previous page)

```
ops.fix(1, 1, 1, 1)
ops.fix(3, 1, 1, 0)

opsv.plot_model()
plt.title('plot_model before defining elements')

ops.geomTransf('Linear', 1)

# columns
ops.element('elasticBeamColumn', 1, 1, 2, Acol, E, IzCol, 1)
ops.element('elasticBeamColumn', 2, 3, 4, Acol, E, IzCol, 1)
# girder
ops.element('elasticBeamColumn', 3, 2, 4, Agir, E, IzGir, 1)

Px = 2.e+3
Wy = -10.e+3
Wx = 0.

Ew = {3: ['-beamUniform', Wy, Wx]}

ops.timeSeries('Constant', 1)
ops.pattern('Plain', 1, 1)
ops.load(2, Px, 0., 0.)

for etag in Ew:
    ops.eleLoad('-ele', etag, '-type', Ew[etag][0], Ew[etag][1],
                Ew[etag][2])

ops.constraints('Transformation')
ops.numberer('RCM')
ops.system('BandGeneral')
ops.test('NormDispIncr', 1.0e-6, 6, 2)
ops.algorithm('Linear')
ops.integrator('LoadControl', 1)
ops.analysis('Static')
ops.analyze(1)

ops.printModel()

opsv.plot_model()
plt.title('plot_model after defining elements')

opsv.plot_loads_2d()

# sfac = 80.

opsv.plot_defo()
# opsv.plot_defo(sfac)
# fmt_interp = {'color': 'blue', 'linestyle': 'solid', 'linewidth': 1.2, 'marker': '.', 'markersize
↪': 6}
# opsv.plot_defo(sfac, fmt_interp=fmt_interp)
```

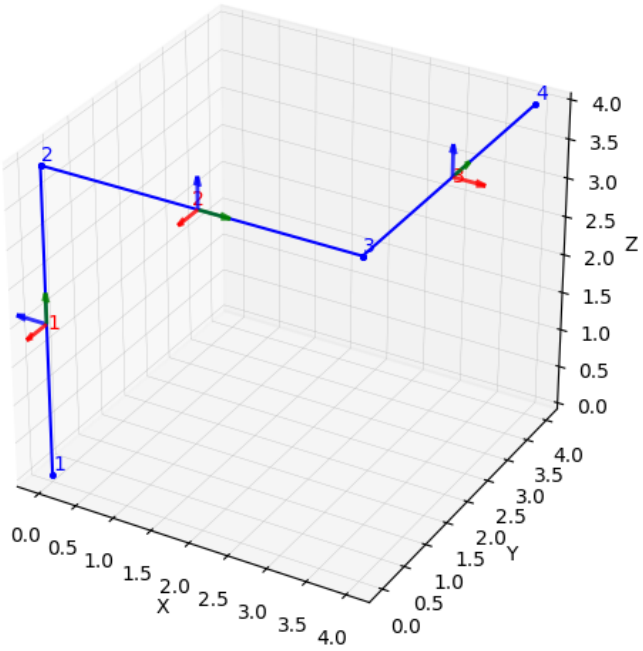
(continues on next page)

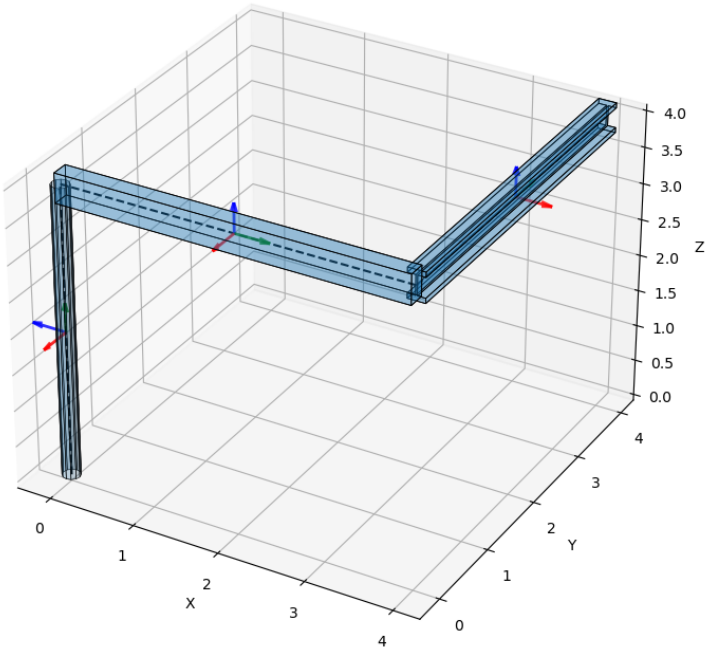
(continued from previous page)

```
75
76  # 4. plot N, V, M forces diagrams
77
78  sfacN, sfacV, sfacM = 5.e-5, 5.e-5, 5.e-5
79
80  opsv.section_force_diagram_2d('N', sfacN)
81  plt.title('Axial force distribution')
82
83  opsv.section_force_diagram_2d('T', sfacV)
84  plt.title('Shear force distribution')
85
86  opsv.section_force_diagram_2d('M', sfacM)
87  plt.title('Bending moment distribution')
88
89  plt.show()
90
91  exit()
```

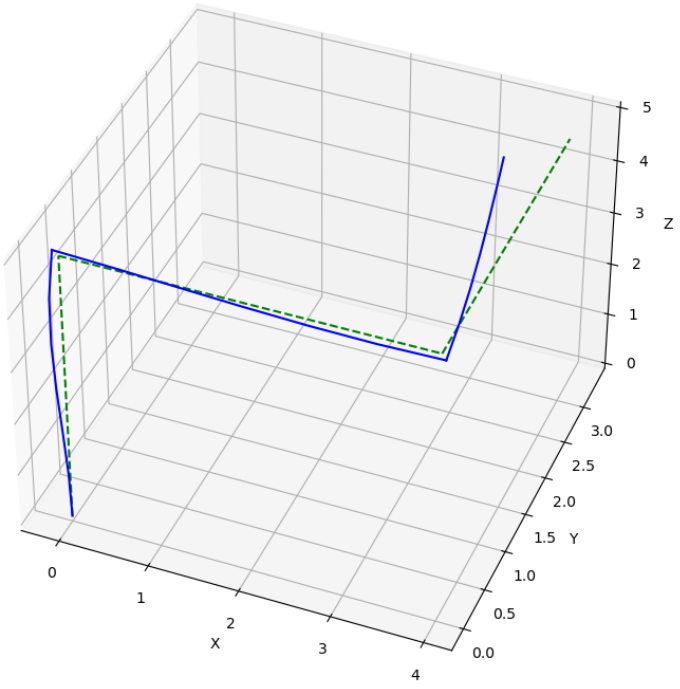
14.2 Statics of a 3d 3-element cantilever beam

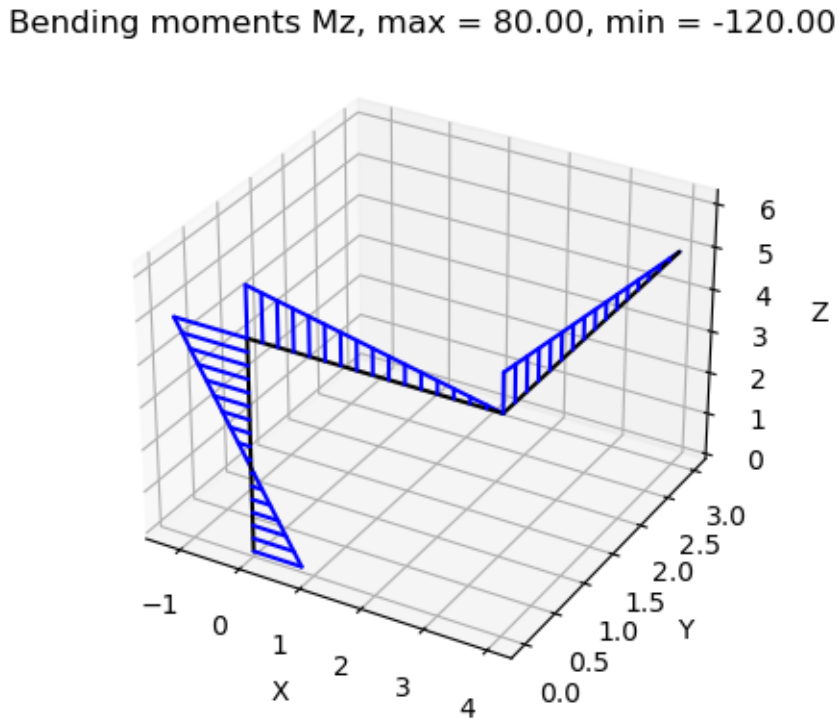
Example `.py` file can be downloaded `here`:

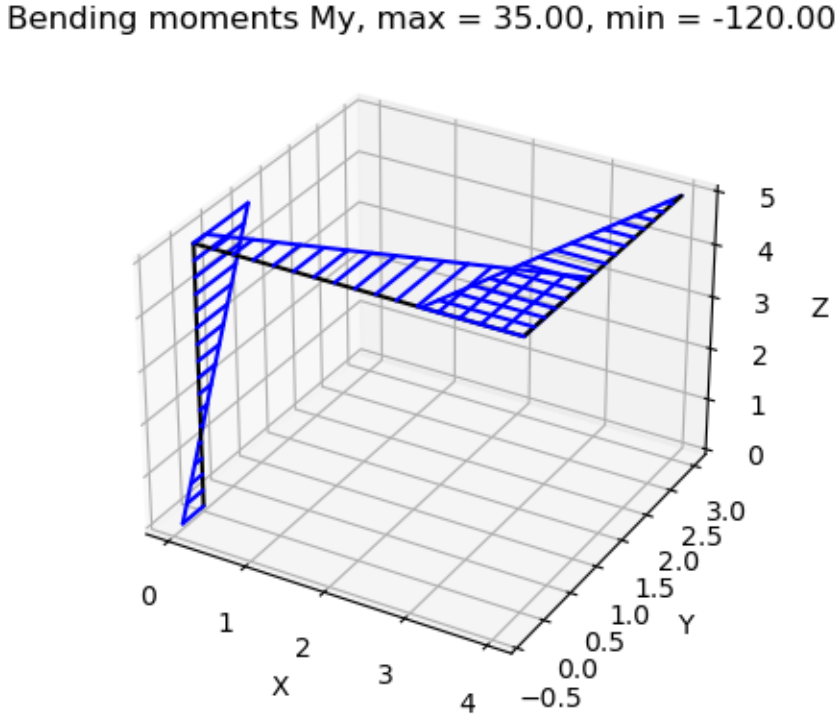




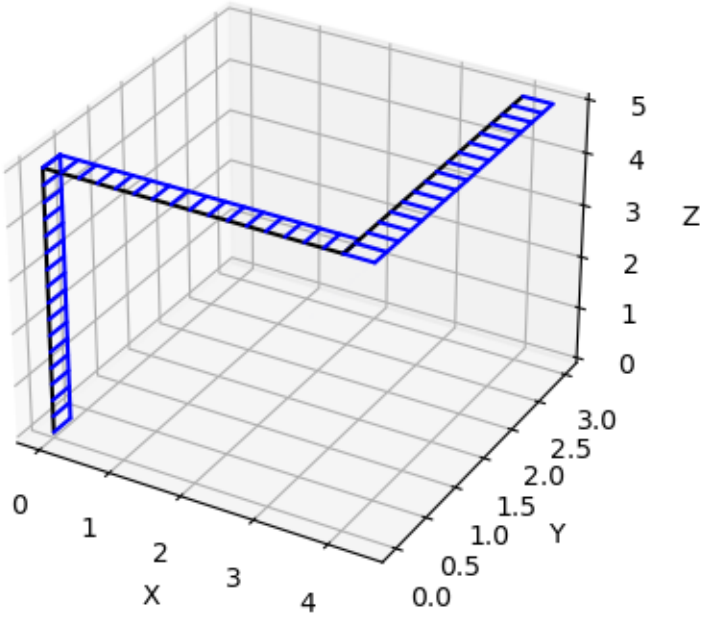
3d 3-element cantilever beam



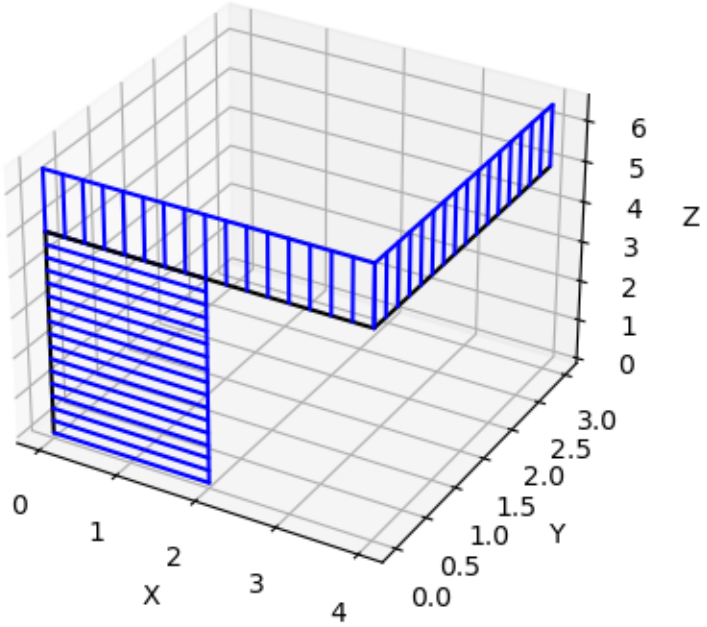

Bending moments Mz, max = 80.00, min = -120.00
6
5
4
3
2
1
0
Z
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Y
−1
0
1
2
3
4
X

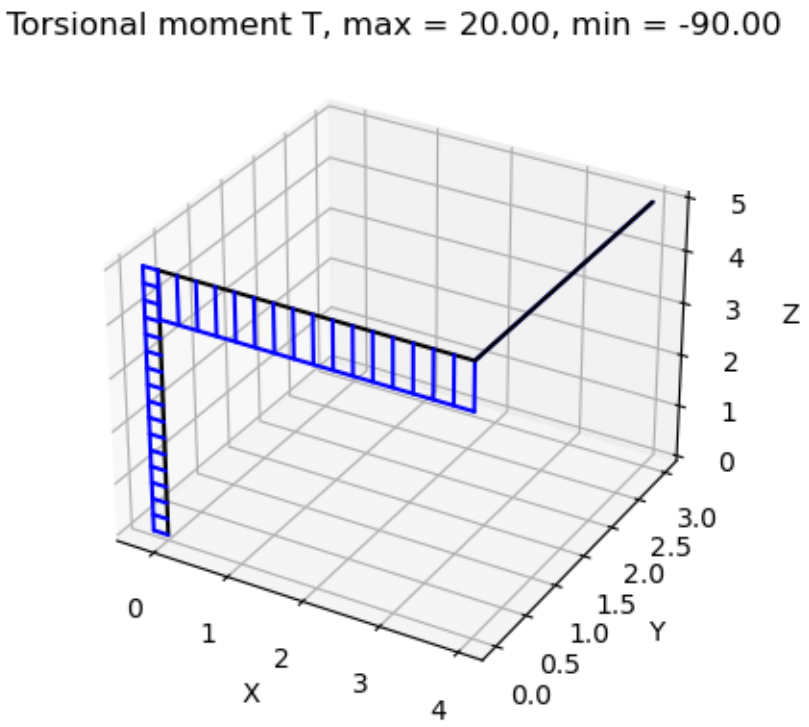

Bending moments My, max = 35.00, min = -120.00
5
4
3
2
1
0
Z
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
Y
0
1
2
3
4
X

Transverse force Vz, max = 40.00, min = -25.00



Transverse force Vy, max = 30.00, min = -40.00






```
import openseespy.opensees as ops
import opsvis as opsv

import matplotlib.pyplot as plt

ops.wipe()

ops.model('basic', '-ndm', 3, '-ndf', 6)

b = 0.2
h = 0.4

A, Iz, Iy, J = 0.04, 0.0010667, 0.0002667, 0.01172

E = 25.0e6
G = 9615384.6

# Lx, Ly, Lz = 4., 3., 5.
Lx, Ly, Lz = 4., 4., 4.

ops.node(1, 0., 0., 0.)
ops.node(2, 0., 0., Lz)
ops.node(3, Lx, 0., Lz)
ops.node(4, Lx, Ly, Lz)

ops.fix(1, 1, 1, 1, 1, 1, 1)

lmass = 200.
```

(continues on next page)

(continued from previous page)

```
29
30 ops.mass(2, lmass, lmass, lmass, 0.001, 0.001, 0.001)
31 ops.mass(3, lmass, lmass, lmass, 0.001, 0.001, 0.001)
32 ops.mass(4, lmass, lmass, lmass, 0.001, 0.001, 0.001)
33
34 gTTagz = 1
35 gTTagx = 2
36 gTTagy = 3
37
38 coordTransf = 'Linear'
39 ops.geomTransf(coordTransf, gTTagz, 0., -1., 0.)
40 ops.geomTransf(coordTransf, gTTagx, 0., -1., 0.)
41 ops.geomTransf(coordTransf, gTTagy, 1., 0., 0.)
42
43 ops.element('elasticBeamColumn', 1, 1, 2, A, E, G, J, Iy, Iz, gTTagz)
44 ops.element('elasticBeamColumn', 2, 2, 3, A, E, G, J, Iy, Iz, gTTagx)
45 ops.element('elasticBeamColumn', 3, 3, 4, A, E, G, J, Iy, Iz, gTTagy)
46
47 Ew = {}
48
49 Px = -4.e1
50 Py = -2.5e1
51 Pz = -3.e1
52
53 ops.timeSeries('Constant', 1)
54 ops.pattern('Plain', 1, 1)
55 ops.load(4, Px, Py, Pz, 0., 0., 0.)
56
57 ops.constraints('Transformation')
58 ops.numberer('RCM')
59 ops.system('BandGeneral')
60 ops.test('NormDispIncr', 1.0e-6, 6, 2)
61 ops.algorithm('Linear')
62 ops.integrator('LoadControl', 1)
63 ops.analysis('Static')
64 ops.analyze(1)
65
66
67 opsv.plot_model()
68
69 sfac = 2.0e0
70
71 # fig_wi_he = 22., 14.
72 fig_wi_he = 30., 20.
73
74 # - 1
75 nep = 9
76 opsv.plot_defo(sfac, nep, az_el=(-68., 39.),
77                fig_wi_he=fig_wi_he, endDispFlag=0)
78
79 plt.title('3d 3-element cantilever beam')
80
```

(continues on next page)

(continued from previous page)

```
# - 2
opsv.plot_defo(sfac, 19, az_el=(6., 30.), fig_wi_he=fig_wi_he)

plt.title('3d 3-element cantilever beam')

# - 3
nfreq = 6
eigValues = ops.eigen(nfreq)

modeNo = 6

sfac = 2.0e1
opsv.plot_mode_shape(modeNo, sfac, 19, az_el=(106., 46.),
                     fig_wi_he=fig_wi_he)
plt.title(f'Mode {modeNo}')

sfacN = 1.e-2
sfacVy = 5.e-2
sfacVz = 1.e-2
sfacMy = 1.e-2
sfacMz = 1.e-2
sfacT = 1.e-2

# plt.figure()
opsv.section_force_diagram_3d('N', sfacN)
plt.title('Axial force N')

opsv.section_force_diagram_3d('Vy', sfacVy)
plt.title('Transverse force Vy')

opsv.section_force_diagram_3d('Vz', sfacVz)
plt.title('Transverse force Vz')

opsv.section_force_diagram_3d('My', sfacMy)
plt.title('Bending moments My')

opsv.section_force_diagram_3d('Mz', sfacMz)
plt.title('Bending moments Mz')

opsv.section_force_diagram_3d('T', sfacT)
plt.title('Torsional moment T')

# just for demonstration,
# the section data below does not match the data in OpenSees model above
# For now it can be source of inconsistency because OpenSees has
# not got functions to return section dimensions.
# A workaround is to have own Python helper functions to reuse data
# specified once
ele_shapes = {1: ['circ', [h]],
              2: ['rect', [b, h]],
              3: ['I', [b, h, b/10., h/6.]]}
opsv.plot_extruded_shapes_3d(ele_shapes, fig_wi_he=fig_wi_he)
```

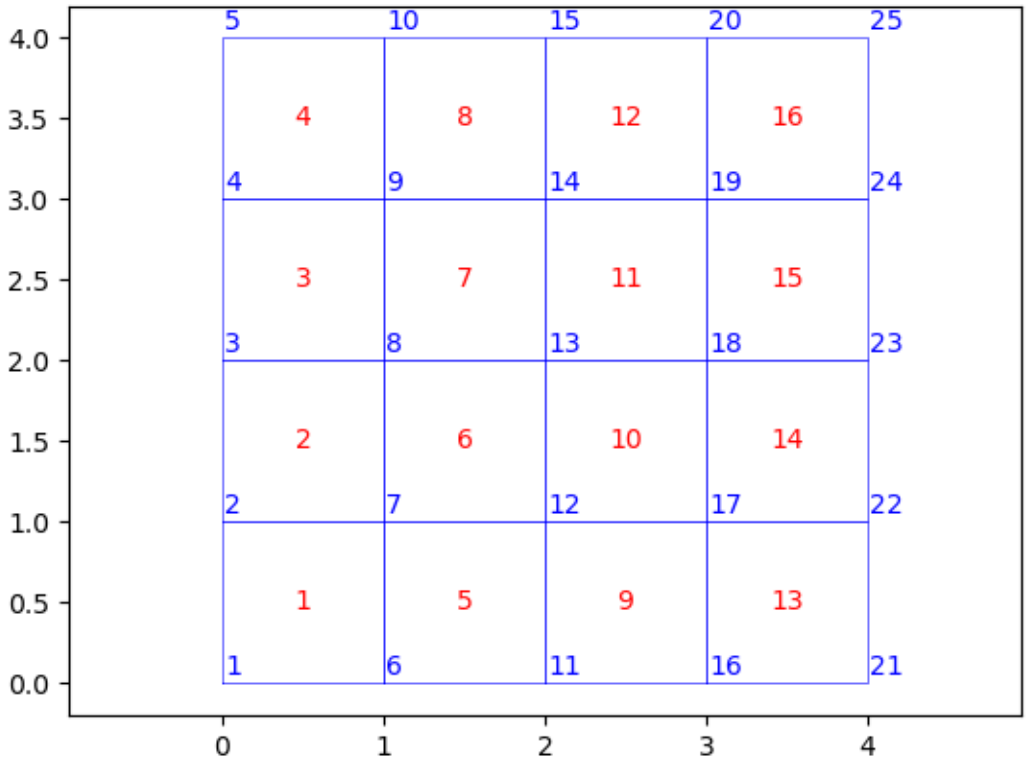
(continues on next page)

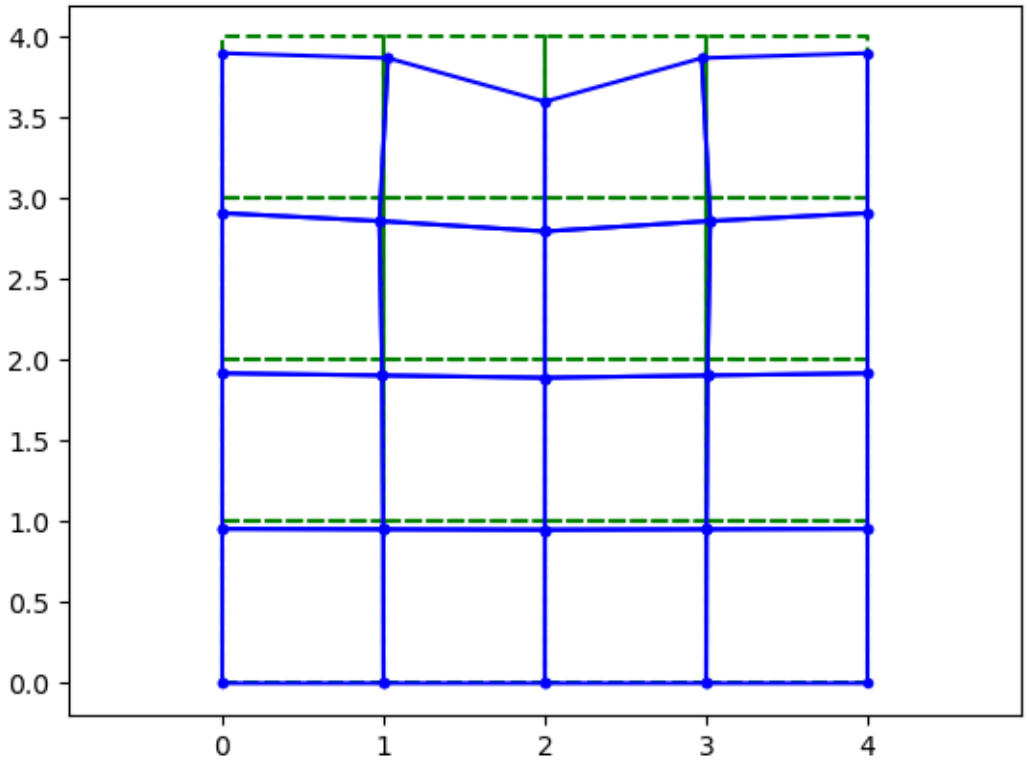
(continued from previous page)

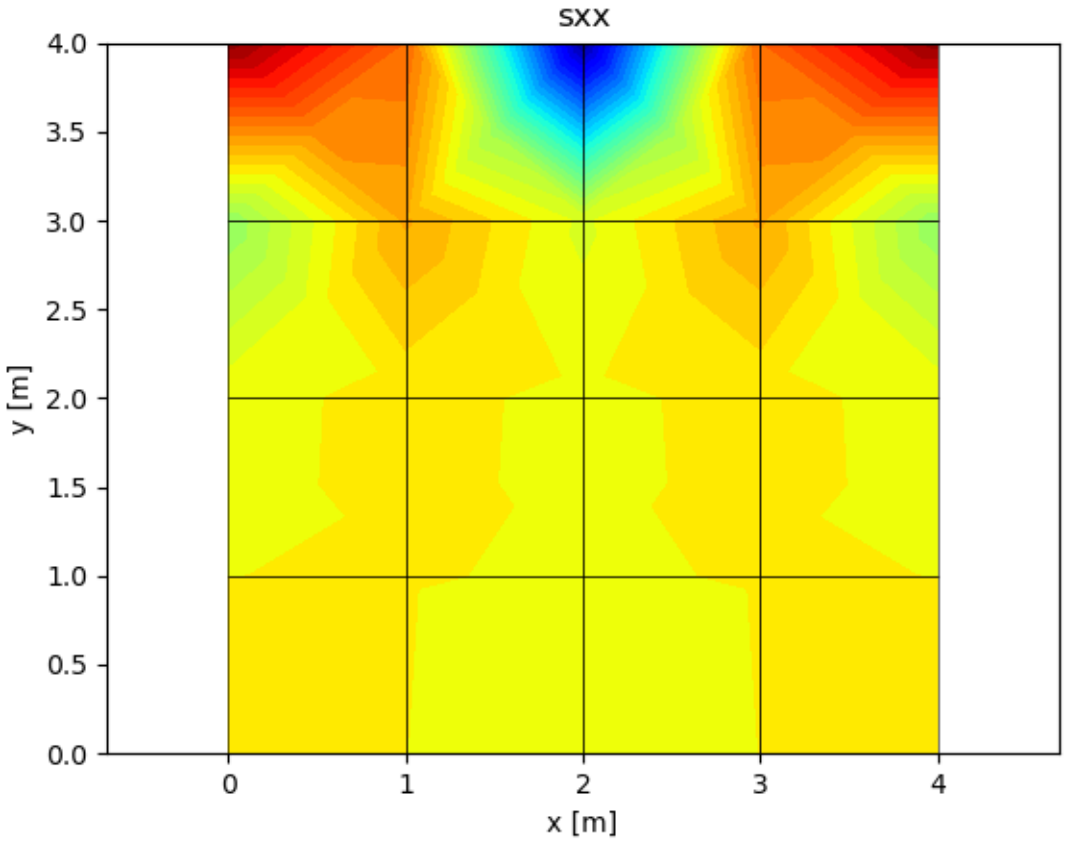
```

plt.show()

exit()
```

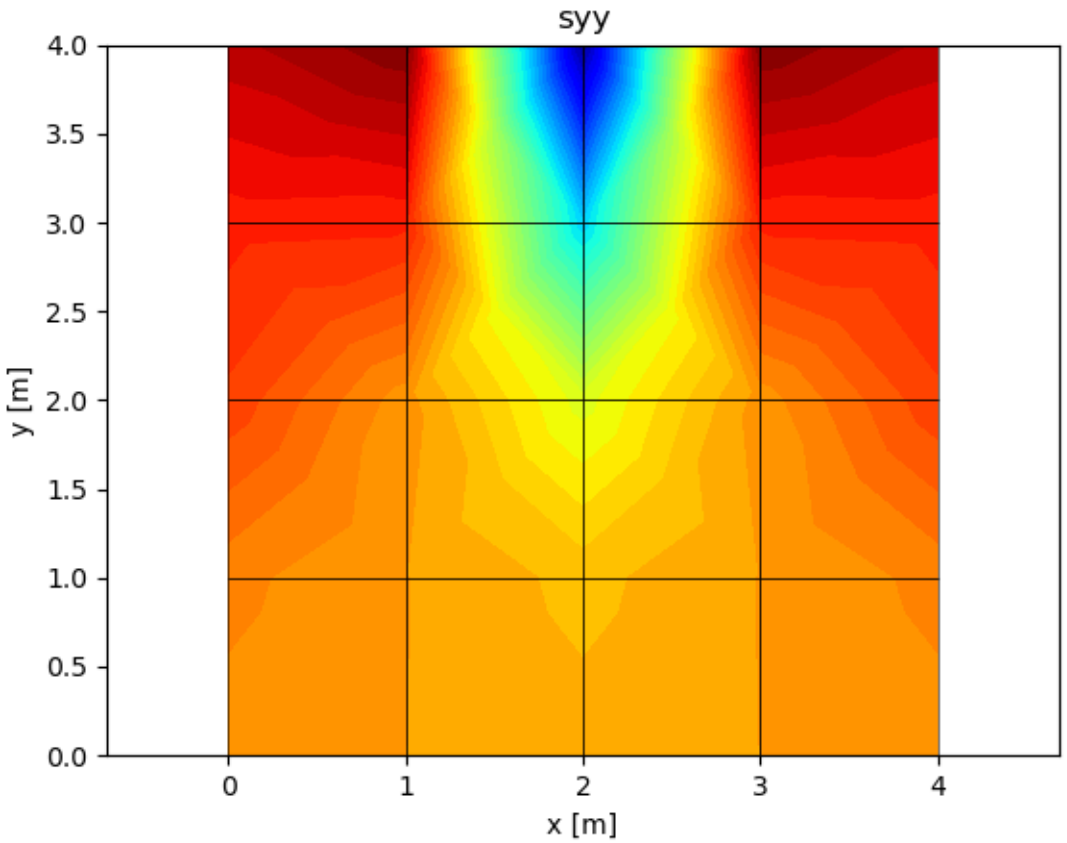
14.3 Plot stress distribution of a plane stress quad model

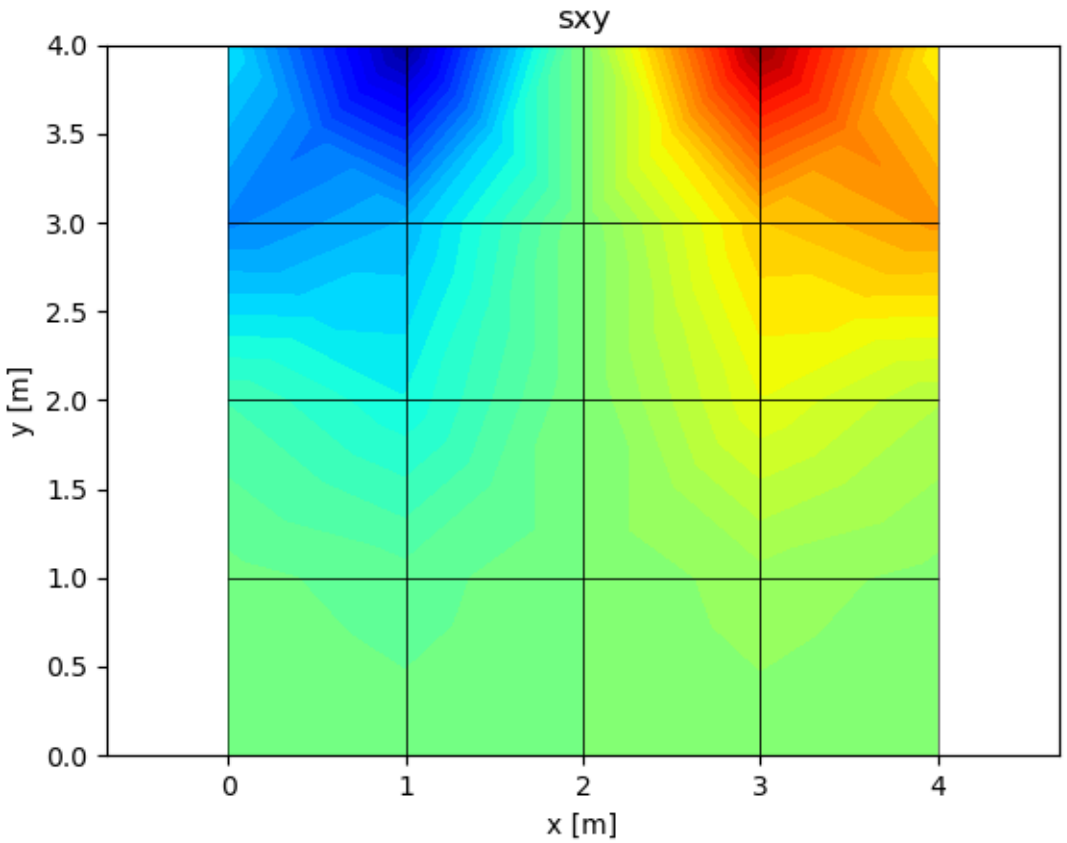
Example `.py` file can be downloaded `here`:

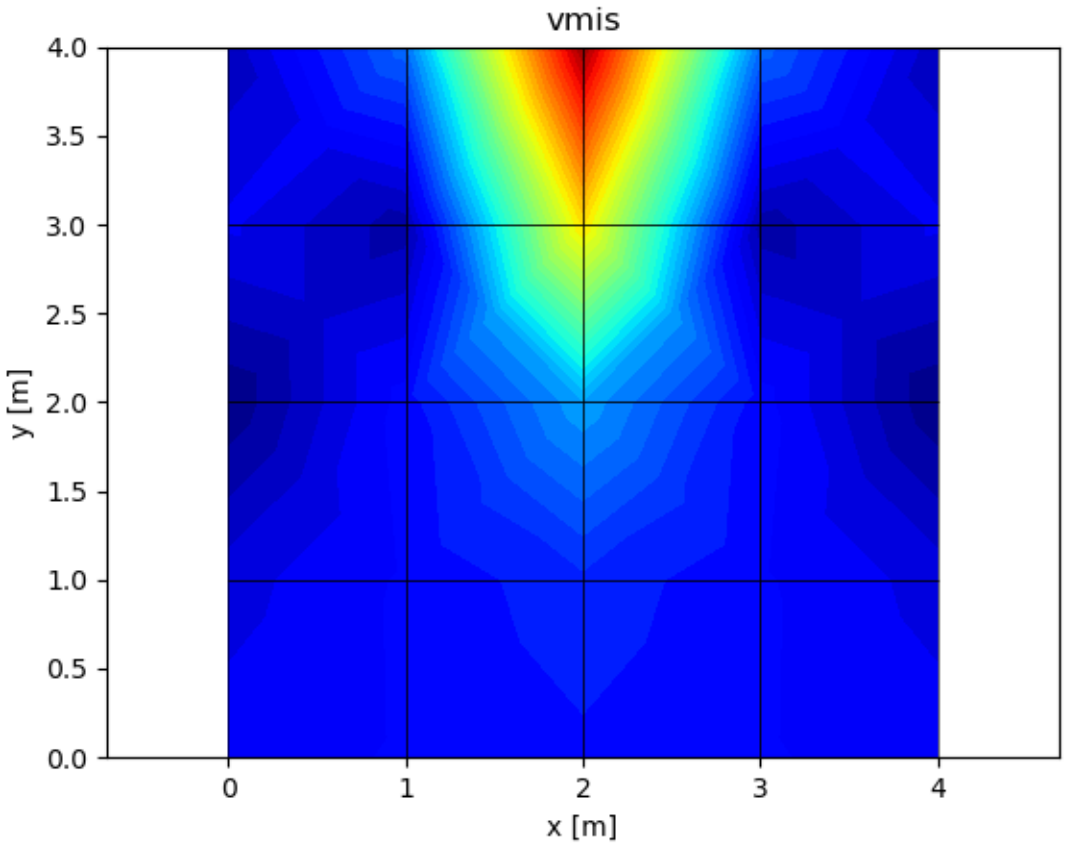



4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4


sxx
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
y [m]
0
1
2
3
4
x [m]


syy
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
y [m]
0
1
2
3
4
x [m]


sxy
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
y [m]
0
1
2
3
4
x [m]




```
import openseespy.opensees as ops
import opsvis as opsv

import matplotlib.pyplot as plt

ops.wipe()
ops.model('basic', '-ndm', 2, '-ndf', 2)
ops.node(1, 0., 0.)
ops.node(2, 0., 1.)
ops.node(3, 0., 2.)
ops.node(4, 0., 3.)
ops.node(5, 0., 4.)
ops.node(6, 1., 0.)
ops.node(7, 1., 1.)
ops.node(8, 1., 2.)
ops.node(9, 1., 3.)
ops.node(10, 1., 4.)
ops.node(11, 2., 0.)
ops.node(12, 2., 1.)
ops.node(13, 2., 2.)
ops.node(14, 2., 3.)
ops.node(15, 2., 4.)
ops.node(16, 3., 0.)
```

(continues on next page)

(continued from previous page)

```
ops.node(17, 3., 1.)
ops.node(18, 3., 2.)
ops.node(19, 3., 3.)
ops.node(20, 3., 4.)
ops.node(21, 4., 0.)
ops.node(22, 4., 1.)
ops.node(23, 4., 2.)
ops.node(24, 4., 3.)
ops.node(25, 4., 4.)

ops.nDMaterial('ElasticIsotropic', 1, 1000, 0.3)

ops.element('quad', 1, 1, 6, 7, 2, 1, 'PlaneStress', 1)
ops.element('quad', 2, 2, 7, 8, 3, 1, 'PlaneStress', 1)
ops.element('quad', 3, 3, 8, 9, 4, 1, 'PlaneStress', 1)
ops.element('quad', 4, 4, 9, 10, 5, 1, 'PlaneStress', 1)
ops.element('quad', 5, 6, 11, 12, 7, 1, 'PlaneStress', 1)
ops.element('quad', 6, 7, 12, 13, 8, 1, 'PlaneStress', 1)
ops.element('quad', 7, 8, 13, 14, 9, 1, 'PlaneStress', 1)
ops.element('quad', 8, 9, 14, 15, 10, 1, 'PlaneStress', 1)
ops.element('quad', 9, 11, 16, 17, 12, 1, 'PlaneStress', 1)
ops.element('quad', 10, 12, 17, 18, 13, 1, 'PlaneStress', 1)
ops.element('quad', 11, 13, 18, 19, 14, 1, 'PlaneStress', 1)
ops.element('quad', 12, 14, 19, 20, 15, 1, 'PlaneStress', 1)
ops.element('quad', 13, 16, 21, 22, 17, 1, 'PlaneStress', 1)
ops.element('quad', 14, 17, 22, 23, 18, 1, 'PlaneStress', 1)
ops.element('quad', 15, 18, 23, 24, 19, 1, 'PlaneStress', 1)
ops.element('quad', 16, 19, 24, 25, 20, 1, 'PlaneStress', 1)

ops.fix(1, 1, 1)
ops.fix(6, 1, 1)
ops.fix(11, 1, 1)
ops.fix(16, 1, 1)
ops.fix(21, 1, 1)

ops.equalDOF(2, 22, 1, 2)
ops.equalDOF(3, 23, 1, 2)
ops.equalDOF(4, 24, 1, 2)
ops.equalDOF(5, 25, 1, 2)

ops.timeSeries('Linear', 1)
ops.pattern('Plain', 1, 1)
ops.load(15, 0., -1.)

ops.analysis('Static')
ops.analyze(1)

# - plot model
opsv.plot_model()
plt.axis('equal')

# plt.figure()
```

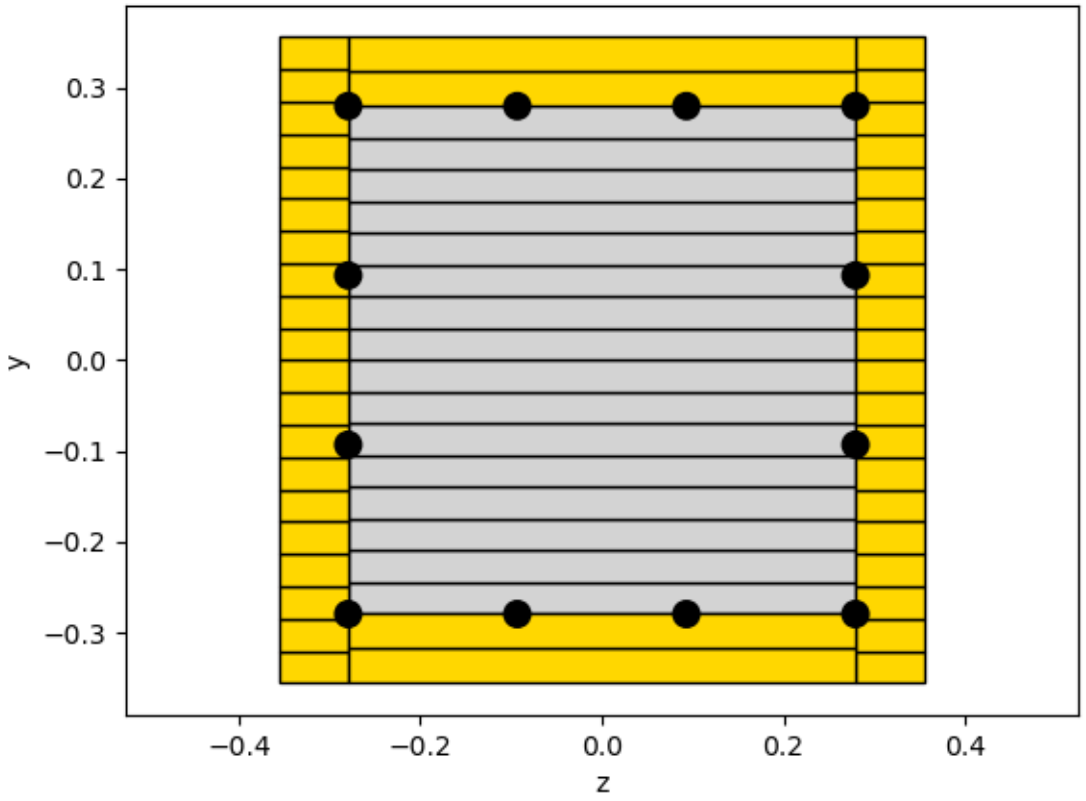
(continues on next page)

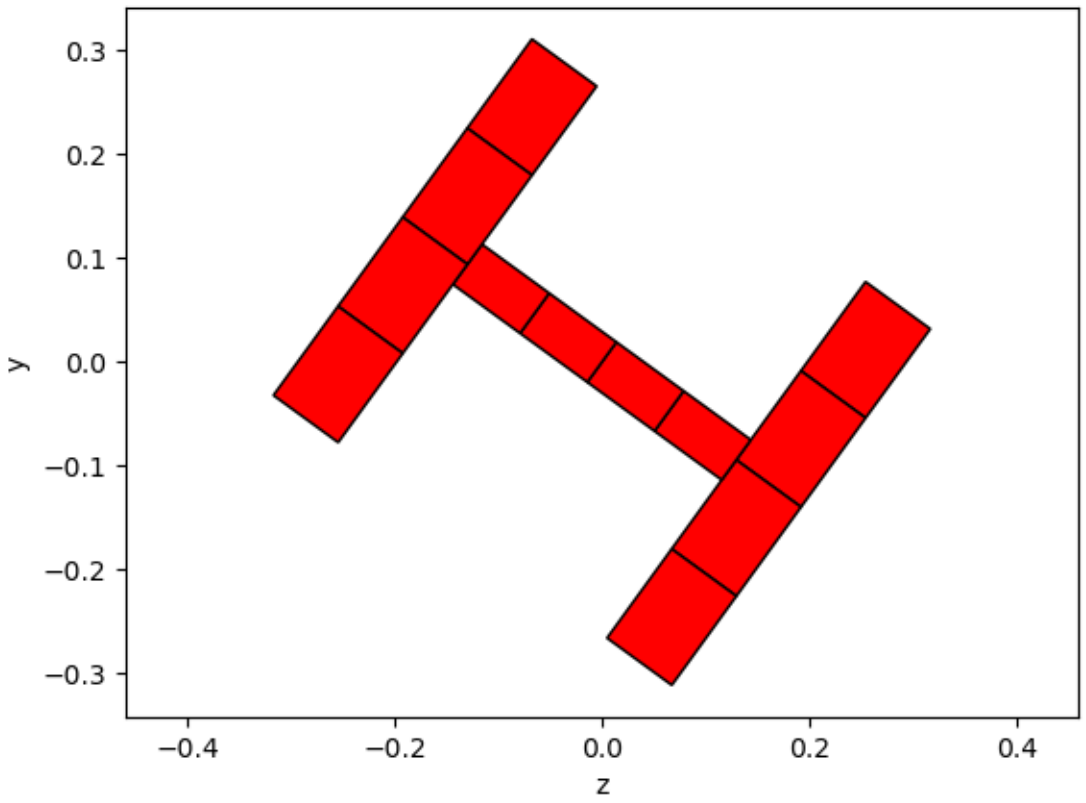
(continued from previous page)

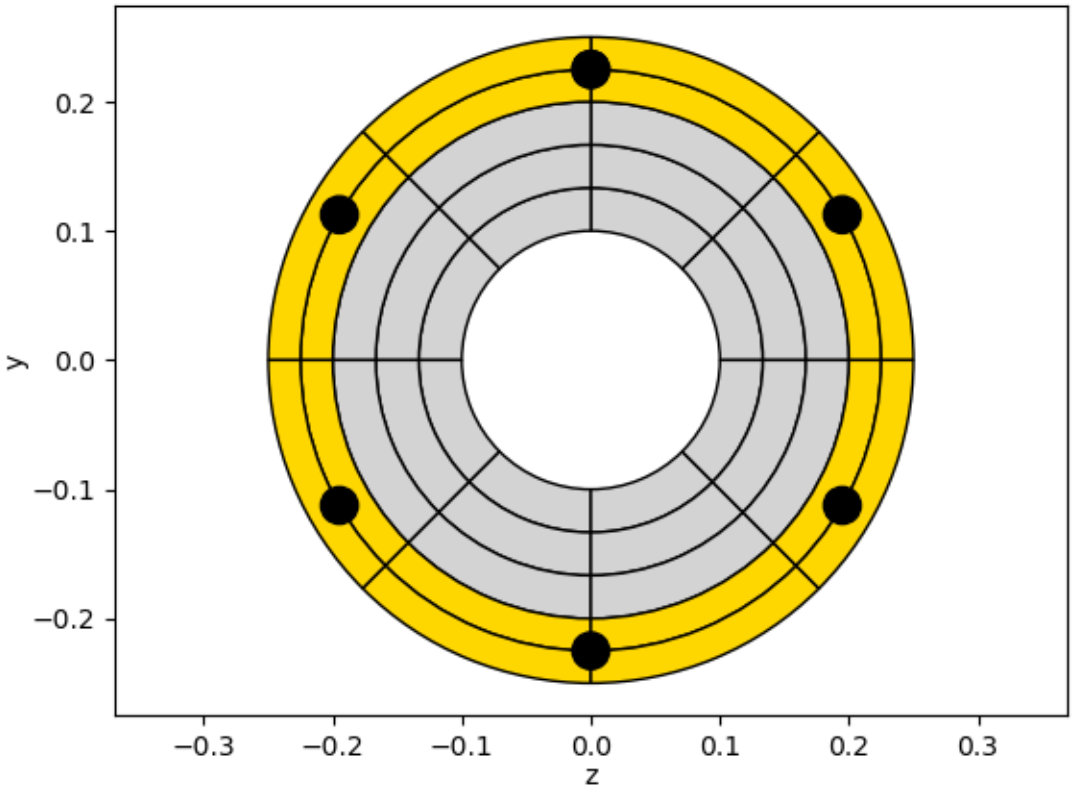
```
opsv.plot_loads_2d()

# - plot deformation
opsv.plot_defo(unDefoFlag=1)
plt.axis('equal')

# get values at OpenSees nodes
sig_out = opsv.sig_out_per_node()

# !!! select from sig_out: e.g. vmises
# j, jstr = 0, 'sxx'
j, jstr = 1, 'syy'
# j, jstr = 2, 'sxy'
# j, jstr = 3, 'vmis'
# j, jstr = 4, 's1'
# j, jstr = 5, 's2'
# j, jstr = 6, 'alfa'

nds_val = sig_out[:, j]

plt.figure()
opsv.plot_stress_2d(nds_val)

plt.show()

exit()
```

14.4 Plot steel and reinforced concrete fiber sections

Example `.py` file can be downloaded here:


0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
y
−0.4
−0.2
0.0
0.2
0.4
z


0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
y
−0.4
−0.2
0.0
0.2
0.4
z




```
import openseespy.opensees as ops
# import opensees as ops  # local compilation
import opsvis as opsv

import matplotlib.pyplot as plt

ops.wipe()
ops.model('basic', '-ndm', 2, '-ndf', 3)  # frame 2D

# 1. rotated steel shape

fib_sec_1 = [['section', 'Fiber', 1, '-GJ', 1.0e6],
             ['patch', 'quad', 1, 4, 1,  0.032, 0.317, -0.311, 0.067, -0.266, 0.005, 0.
↪077, 0.254],
             ['patch', 'quad', 1, 1, 4,  -0.075, 0.144, -0.114, 0.116, 0.075, -0.144, 0.
↪114, -0.116],
             ['patch', 'quad', 1, 4, 1,  0.266, -0.005,  -0.077, -0.254,  -0.032, -0.317,
↪  0.311, -0.067]
             ]

# fib_sec_1 list can be used both for plotting and OpenSees commands defining
# the actual fiber section in the OpenSees domain. Normally you would have to
# use regular section('Fiber', ...), fiber(), layer(), patch() commands with
```

(continues on next page)

(continued from previous page)

```
# the same data to define the fiber section. However do not use both
# ways in the same model.

# opsv.fib_sec_list_to_cmds(fib_sec_1)

# 2. RC section

Bcol = 0.711
Hcol = Bcol

c = 0.076  # cover

y1col = Hcol/2.0
z1col = Bcol/2.0

y2col = 0.5*(Hcol-2*c)/3.0

nFibZ = 1
nFib = 20
nFibCover, nFibCore = 2, 16
As9 = 0.0006446

fib_sec_2 = [['section', 'Fiber', 3, '-GJ', 1.0e6],
             ['patch', 'rect', 2, nFibCore, nFibZ, c-y1col, c-z1col, y1col-c, z1col-c],
             ['patch', 'rect', 3, nFib, nFibZ, -y1col, -z1col, y1col, c-z1col],
             ['patch', 'rect', 3, nFib, nFibZ, -y1col, z1col-c, y1col, z1col],
             ['patch', 'rect', 3, nFibCover, nFibZ, -y1col, c-z1col, c-y1col, z1col-c],
             ['patch', 'rect', 3, nFibCover, nFibZ, y1col-c, c-z1col, y1col, z1col-c],
             ['layer', 'straight', 4, 4, As9, y1col-c, z1col-c, y1col-c, c-z1col],
             ['layer', 'straight', 4, 2, As9, y2col, z1col-c, y2col, c-z1col],
             ['layer', 'straight', 4, 2, As9, -y2col, z1col-c, -y2col, c-z1col],
             ['layer', 'straight', 4, 4, As9, c-y1col, z1col-c, c-y1col, c-z1col]]


# opsv.fib_sec_list_to_cmds(fib_sec_2)

matcolor = ['r', 'lightgrey', 'gold', 'w', 'w', 'w']
opsv.plot_fiber_section(fib_sec_1, matcolor=matcolor)
plt.axis('equal')
# plt.savefig('fibsec_wshape.png')

matcolor = ['r', 'lightgrey', 'gold', 'w', 'w', 'w']
opsv.plot_fiber_section(fib_sec_2, matcolor=matcolor)
plt.axis('equal')
# plt.savefig('fibsec_rc.png')

# 3. circular cross-section
nc1, nr1 = 8, 3
nc2, nr2 = 8, 2
ri1, re1 = 0.1, 0.2
ri2, re2 = 0.2, 0.25
a_beg, a_end = 0., 360.
```

(continues on next page)

(continued from previous page)

```
rbar3 = 0.225
a_beg2, a_end2 = 0., 360.

fib_sec_3 = [['section', 'Fiber', 1, '-GJ', 1.0e6],
             ['patch', 'circ', 2, nc1, nr1, 0., 0., ri1, re1, a_beg, a_end],
             ['patch', 'circ', 3, nc2, nr2, 0., 0., ri2, re2, a_beg, a_end],
             ['layer', 'circ', 4, 6, As9, 0., 0., rbar3, a_beg2, a_end2],
             ]


matcolor = ['r', 'lightgrey', 'gold', 'w', 'w', 'w']
opsv.plot_fiber_section(fib_sec_3, matcolor=matcolor)
plt.axis('equal')
# plt.savefig('fibsec_circ.png')

plt.show()
```

14.5 Animation of dynamic analysis and mode shapes of a 2d Portal Frame

Example `.py` file can be downloaded `here`:

If you do not see the animation running, depending on Python environment, you will have to set a matplotlib backand which supports animation.

For example to see the matplotlib animations in the Spyder Python editor (which should be installed with the Anaconda platform), in the IPython console run `%matplotlib qt`. Then comment/uncomment one of the two animations.

```
import openseespy.opensees as ops
import opsvis as opsv

import numpy as np
import matplotlib.pyplot as plt

# input_parameters = (3.8, 50., 100.)
# input_parameters = (53.5767, 50., 100.)
input_parameters = (20.8, 300., 8.)
# input_parameters = (70.0, 500., 2.)

pf, sfac_a, tkt = input_parameters

ops.wipe()
ops.model('basic', '-ndm', 2, '-ndf', 3)  # frame 2D

colL, girL = 4., 6.

Acol, Agir = 0.06, 0.06
IzCol, IzGir = 0.0002, 0.0002

E = 3.2e10
```

(continues on next page)

(continued from previous page)

```
rho = 2400.
muCol = rho * Acol
muGir = rho * Agir
massCol = ['-mass', muCol, '-cMass']
massGir = ['-mass', muGir, '-cMass']

ops.node(0, 0.0, 0.0)
ops.node(1, 0.0, 2.0)
ops.node(2, 0.0, 4.0)
ops.node(3, 3.0, 4.0)
ops.node(4, 6.0, 4.0)
ops.node(5, 6.0, 2.0)
ops.node(6, 6.0, 0.0)

ops.fix(0, 1, 1, 1)
ops.fix(6, 1, 1, 0)

gTTag = 1
ops.geomTransf('Linear', gTTag)

# 1st column
ops.element('elasticBeamColumn', 1, 0, 1, Acol, E, IzCol, gTTag, *massCol)
ops.element('elasticBeamColumn', 2, 1, 2, Acol, E, IzCol, gTTag, *massCol)
# girder
ops.element('elasticBeamColumn', 3, 2, 3, Agir, E, IzGir, gTTag, *massGir)
ops.element('elasticBeamColumn', 4, 3, 4, Agir, E, IzGir, gTTag, *massGir)
# 2nd column
ops.element('elasticBeamColumn', 5, 4, 5, Acol, E, IzCol, gTTag, *massCol)
ops.element('elasticBeamColumn', 6, 5, 6, Acol, E, IzCol, gTTag, *massCol)

t0 = 0.
tk = 1.
Tp = 1/pf
P0 = 15000.
dt = 0.002
n_steps = int((tk-t0)/dt)

tsTag = 1
ops.timeSeries('Trig', tsTag, t0, tk, Tp, '-factor', P0)

patTag = 1
ops.pattern('Plain', patTag, tsTag)
ops.load(1, 1., 0., 0.)

ops.constraints('Transformation')
ops.numberer('RCM')
ops.test('NormDispIncr', 1.0e-6, 10, 1)
ops.algorithm('Linear')
ops.system('ProfileSPD')
ops.integrator('Newmark', 0.5, 0.25)
ops.analysis('Transient')

```

(continues on next page)

(continued from previous page)

```
el_tags = ops.getEleTags()

nels = len(el_tags)

Eds = np.zeros((n_steps, nels, 6))
timeV = np.zeros(n_steps)

# transient analysis loop and collecting the data
for step in range(n_steps):
    ops.analyze(1, dt)
    timeV[step] = ops.getTime()
    # collect disp for element nodes
    for el_i, ele_tag in enumerate(el_tags):
        nd1, nd2 = ops.eleNodes(ele_tag)
        Eds[step, el_i, :] = [ops.nodeDisp(nd1)[0],
                              ops.nodeDisp(nd1)[1],
                              ops.nodeDisp(nd1)[2],
                              ops.nodeDisp(nd2)[0],
                              ops.nodeDisp(nd2)[1],
                              ops.nodeDisp(nd2)[2]]

fmt_defo = {'color': 'blue', 'linestyle': 'solid', 'linewidth': 3.0,
            'marker': '', 'markersize': 6}

# 1. animate the deformated shape
anim = opsv.anim_defo(Eds, timeV, sfac_a, fmt_defo=fmt_defo,
                      xlim=[-1, 7], ylim=[-1, 5], fig_wi_he=(30., 22.))

plt.show()

# 2. after closing the window, animate the specified mode shape
eigVals = ops.eigen(5)

modeNo = 2  # specify which mode to animate
f_modeNo = np.sqrt(eigVals[modeNo-1])/(2*np.pi)  # i-th natural frequency

anim = opsv.anim_mode(modeNo, fmt_defo=fmt_defo,
                      xlim=[-1, 7], ylim=[-1, 5], fig_wi_he=(30., 22.))
plt.title(f'Mode {modeNo}, f_{modeNo}: {f_modeNo:.3f} Hz')

plt.show()
```

1. *2d Portal Frame*
2. *Statics of a 3d 3-element cantilever beam*
3. *Plot stress distribution of a plane stress quad model*
4. *Plot steel and reinforced concrete fiber sections*
5. *Animation of dynamic analysis and mode shapes of a 2d Portal Frame*

Warning - Incompatible changes!

Starting from Opsvis ver. 1.0.1 (May 2022):

1. the `plot_supports_and_loads_2d` function is removed and instead `plot_loads_2d` is available. The model supports can now be shown in the `plot_model` function. The additiona function argument `node_supports` (default is True) can be switched off (e.g. `plot_model(node_supports=False)`).
2. the `plot_mesh_with_ips_2d()` function is removed.
3. See and use the updated example `.py` files (*Examples*).
4. Now the opsvis plotting functions use the Python dictionary (instead of the string) for the maptplotlib line formatting. Example: Use `fmt_model = {'color': 'blue', 'linestyle': 'solid', 'linewidth': 1.2, 'marker': '.', 'markersize': 6}` instead of the previous `fmt_defo = 'b-'` string format. Also the `lw` (linewidth) arguments are removed from the plotting functions as they can be defined in the `fmt_*` dictionary. This feature has been implemented as suggested by mbbatukan.

Opsvis is an OpenSeesPy postprocessing and visualization module written by Seweryn Kokot (Opole University of Technology, Poland).

For OpenSeesPy documentation click the following link: OpenSeesPy documentation

Opsvis can be mainly useful for students when learning the fundamentals of structural analysis (interpolated deformation of frame structures (static images or animations), section force distribution of frame structures, stress distribution in triangle, quadrilateral 2d elements, orientation of frame members in 3d space, fibers of a cross section, static and animated eigenvalue mode shapes etc.). This way, we can lower the bar in teaching and learning OpenSees at earlier years of civil engineering studies. However the visualization features for OpenSees can also be helpful for research studies.

Opsvis offers the following plots:

- interpolated deformation of frame structures,
- stresses of triangular and (four, eight and nine-node) quadrilateral 2d elements (calculation of Huber-Mieses-Hencky equivalent stress, principal stresses),
- fibers of cross-sections,
- models with extruded cross sections
- animation of deformation (from time history analysis) and mode shapes.

CHAPTER

FIFTEEN

INSTALLATION

```
pip install opsvis
```

Note the name of the PyPi package is without the underscore _.

CHAPTER

SIXTEEN

USAGE

To use Opsvis in OpenSeesPy scripts, your `.py` file should start as follows:

```
import openseespy.opensees as ops
import opsvis as opsv
import matplotlib.pyplot as plt

# ... your OpenSeesPy model and analysis commands ...
opsv.plot_model()
sfac = opsv.plot_defo()
```

CHAPTER

SEVENTEEN

COMMANDS

The main commands related to various aspects of OpenSees model visualization are as follows:

1. *plot_model*
2. *plot_defo*
3. *plot_loads_2d*
4. *plot_mode_shape*
5. *section_force_diagram_2d*
6. *section_force_diagram_3d*
7. *plot_stress_2d*
8. *plot_extruded_model_rect_section_3d*
9. *anim_defo*
10. *anim_mode*
11. *plot_fiber_section*

Helper functions include:

1. *fib_sec_list_to_cmds*
2. *sig_out_per_node*

For examples go to: *Examples*.

CHAPTER

EIGHTEEN

NOTES:

- matplotlib's `plt.axis('equal')` does not work for 3d plots therefore right angles are not guaranteed to be 90 degrees on the plots
- `plot_fiber_section` is inspired by Matlab `plotSection.zip` written by D. Vamvatsikos available at http://users.ntua.gr/divamva/software.html