PyFEM – Finite Elements with Python

PyFEM is a finite element set of libraries for multiple types of analysis including static equilibrium, seepage and hydro-mechanical analysis.

Descriptions for each library follows.

# pyfem.mesh library

Pyfem.mesh is the library for creating structured meshes. Volume, area or line blocks can be discretized in a variety of shapes which are compatible with vtk format. Shapes generated using this library are described below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Shape name** | **Number Id** | **Description** | **Interpolation** |
| LIN2 | 3 | Line shape with 2 nodes | Linear |
| LIN3 | 21 | Line shape with 3 nodes | Quadratic |
| TRI3 | 5 | Triangular shape with 3 nodes | Linear |
| TRI6 | 22 | Triangular shape with 6 nodes | Quadratic |
| QUAD4 | 9 | Quadrilateral shape with 4 nodes | Linear |
| QUAD8 | 23 | Quadrilateral shape with 8 nodes | Quadratic |
| TET4 | 10 | Tetrahedral shape with 4 nodes | Linear |
| TET10 | 24 | Tetrahedral shape with 10 nodes | Quadratic |
| HEX8 | 12 | Hexahedral shape with 8 nodes | Linear |
| HEX20 | 25 | Hexahedral shape with 20 nodes | Quadratic |

## Class <Block>

Represent the base class for specific types of blocks. This class should not be instantiated directly. Derived classes inherit the methods described in this item.

### Methods:

|  |  |  |
| --- | --- | --- |
| **Method** | **Returns** | **Description** |
| move(V) | None | Modifies block internal coordinates by applying a displacement according to V vector.  The V vector can be a list of two or three values [dx, dy, dz] where dx, dy and dz represent displacements along x, y, and z axes. |
| set\_linear(C) | None | Sets the resulting shapes as linear shapes. |
| set\_quadratic(C) | None | Sets the resulting shapes as quadratic shapes. |
| set\_cubic(C) | None | Sets the resulting shapes as cubic shapes. |

## Class <BlockLine>

Represent a line block for 2D and 3D spaces. This class inherits the methods from class Block.

### Methods:

|  |  |  |
| --- | --- | --- |
| **Method** | **Returns** | **Description** |
| \_\_init\_\_() | None | Initializes the node object |
| set\_coords(C) | None | Sets the block coordinates according to C matrix. At least two point coordinates are needed.  The C matrix can be a list of lists with each sublist representing a coordinate point. Alternatively, the C matrix can be a numpy matrix. |
| set\_divisions(n) | None | Sets the number of divisions for this block. |
| copy() | BlockLine | Returns a hard copy of the object. |

## Class <Block2D>

Represent a 2D block to represent a quadrilateral region. It should be used for 2D meshes. This class inherits the methods from class Block.

### Methods:

|  |  |  |
| --- | --- | --- |
| **Method** | **Returns** | **Description** |
| \_\_init\_\_() | None | Initializes the node object |
| set\_coords(C) | None | Sets the block coordinates according to C matrix. At least four point coordinates are needed.  The C matrix can be a list of lists with each sublist representing a coordinate point. Alternatively, the C matrix can be a numpy matrix. |
| set\_nx(nx) | None | Sets the number of divisions along local x axis. |
| set\_ny(ny) | None | Sets the number of divisions along local y axis. |
| copy() | Block2D | Returns a hard copy of the object. |

## Class <Block3D>

Represent a 3D block to represent a cubic region. It should be used for 3D meshes. This class inherits the methods from class Block.

### Methods:

|  |  |  |
| --- | --- | --- |
| **Method** | **Returns** | **Description** |
| \_\_init\_\_() | None | Initializes the node object |
| set\_coords(C) | None | Sets the block coordinates according to C matrix. At least four point coordinates are needed.  The C matrix can be a list of lists with each sublist representing a coordinate point. Alternatively, the C matrix can be a numpy matrix. |
| set\_nx(nx) | None | Sets the number of divisions along local x axis. |
| set\_ny(ny) | None | Sets the number of divisions along local y axis. |
| set\_nz(nz) | None | Sets the number of divisions along local z axis. |
| copy() | Block3D | Returns a hard copy of the object. |

## Class <Mesh>

Represent a mesh object for 2D and 3D spaces.

**Methods:**

|  |  |  |
| --- | --- | --- |
| **Method** | **Returns** | **Description** |
| \_\_init\_\_() | None | Initializes the node object |
| set\_ndim(ndim) | None | Sets the mesh space dimension (2 or 3) according to ndim. |
| add\_block(ablock) | None | Adds a block object ablock to an internal block collection. |
| load\_file(filename) | None | Loads a mesh file filename in vtk format. |
| set\_verbose(opt) | None | Sets if the objects outputs extra information during mesh generation according to the logical value opt. |
| generate() | None | Sets the number of divisions along local z axis. |
| write\_file(filename) | None | Writes a file named filename with the mesh information in vtk format. |

# pyfem.fem library

Pyfem.fem is the main library that contains basic classes for finite element analysis. The classes defined in this library are Domain, Element, Face, Node, Ip, CollectionElem, CollectionFace, CollectionNode and CollectionIp.

## Class <Ip>

Represent an integration point inside a finite element. Integration point objects are contained inside Element objects.

### Creation:

ip = Ip() *# Creates an Ip object*

### Properties:

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **In/Out** | **Description** |
| id | int | read only | Represents the integration point Id based on the Ip list from the contained element. |
| r | float | read only | Natural coordinate based on the element’s local coordinates. |
| s | float | read only | Natural coordinate based on the element’s local coordinates. |
| t | float | read only | Natural coordinate based on the element’s local coordinates. |
| x | float | read only | Global coordinate x. |
| y | float | read only | Global coordinate y. |
| z | float | read only | Global coordinate z. |
| mat\_model | MatModel | read-only | Material constitutive model for the integration point. |

## Class <Node>

**Properties:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **In/Out** | **Description** |
| id | int | read only | Represents the node Id based on the nodes list of the container Domain object. |
| x | float | read only | Global x coordinate. |
| y | float | read only | Global y coordinate. |
| z | float | read only | Global z coordinate. |
| tag | str | read-write | Label for the node |

### Methods:

|  |  |  |
| --- | --- | --- |
| **Method** | **Returns** | **Description** |
| \_\_init\_\_() | None | Initializes the node object |
| set\_brys(brys) | None | Sets the nodal boundary conditions. The parameter brys should be a dictionary object containing the respective boundary conditions keys and boundary conditions values. |

## Class <CollectionNode>:

Represents an iterable collection of Node objects.

### Creation:

nodes\_collection = CollNode()

### Properties:

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **In/Out** | **Description** |
| max\_x | float | read only | Maximum x coordinate in collection. |
| max\_y | float | read only | Maximum y coordinate in collection. |
| max\_z | float | read only | Maximum z coordinate in collection. |
| min\_x | float | read only | Minimum x coordinate in collection. |
| min\_y | float | read only | Minimum y coordinate in collection. |
| min\_z | float | read only | Minimum z coordinate in collection. |

### Methods:

|  |  |  |
| --- | --- | --- |
| **Method** | **Output** | **Description** |
| \_\_getitem\_\_(index) | None | Returns a node object from the collection according to the index value using the operator []. |
| append(node) | None | Adds a node object at the end of the collection. |
| extend(nodes\_collection) | None | Extends the collection with another collhection. |
| with\_id(id0,id1,…) | CollectionNode | Filters the CollectionNode object with nodes with the corresponding ids id0, id1, and so on. |
| with\_x(x0,x1,…) | CollectionNode | Filters the CollectionNode object with all nodes with x coordinate equal to x0 or x1 and so on. |
| with\_y(y0,y1,…) | CollectionNode | Analogous to with\_x function. |
| with\_z(z0,z1,…) | CollectionNode | Analogous to with\_x function. |
| with\_x\_in\_interval(x0,x1) | CollectionNode | Filters the CollectionNode object with all nodes with x coordinate inside the closed interval [x0,x1]. |
| with\_y\_in\_interval(y0,y1) | CollectionNode | Analogous to with\_x\_**in**\_interval function. |
| with\_x\_in\_interval(z0,z1) | CollectionNode | Analogous to with\_x\_**in**\_interval function. |
| sub(attr=at1) | CollectionNode | Filters the CollectionNode object returning all nodes that match the specified attribute attr (i.e. x, y, z, id, tag, etc.) with the given value at1. |
| set\_brys(brys) | None | Sets the Node boundary conditions for all Node objects in the collection. The parameter brys should be a dict object containing the respective boundary conditions keys and boundary conditions values. |
| clear\_brys(brys) | None | Clears all boundary conditions from all Node objects in the collection. |

### Example:

domain = Domain() *# Creates a Domain object*

domain.load\_mesh(‘my\_file.vtk’) *# Load mesh information from file*

top\_nodes = domain.nodes.with\_z(10.0) *# Filters all nodes with according to z coordinate*

top\_nodes.set\_bry(uz = 0.0) *# Set boundary conditions* ***in*** *top\_nodes*

## Class <Element>:

### Properties:

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **In/Out** | **Description** |
| id | int | read-only | Represents the element Id based on the element list of the container Domain object. |
| shape\_type | int | read-only | Code that identifies the shape of the element |
| tag | str | read-write | Label for the element |
| nodes | CollectionNode | read-only | Collection with all element nodes |
| elem\_model | ElemModel | read-only | Analysis model for the element. It defines the functionality of the element and must be compatible with the solver to be used. |

### Methods:

|  |  |  |
| --- | --- | --- |
| **Method** | **Returns** | **Description** |
| set\_elem\_model(elem\_model) | None | Sets the analysis model for the element. |

## Class <CollectionElem>:

Represents a collection of Element objects.

### Properties:

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **In/Out** | **Description** |
| nodes | CollectionNode | read-only | Collection of nodes containing all elements nodes from the CollectionElem object. |

### Methods:

|  |  |  |
| --- | --- | --- |
| **Method** | **Returns** | **Description** |
| \_\_getitem\_\_(index) | None | Returns an element object from the collection according to the index value using the operator []. |
| append(elem) | None | Adds an element object at the end of the collection. |
| extend(elem\_collection) | None | Extends the collection with another collection of elements. |
| with\_id(id0,id1,…) | CollectionNode | Filters the collection object returning a new collection with elements with the corresponding ids id0, id1, and so on. |
| With\_tag(tag0,tag1,…) | CollectionElem | Filters the collection object returning a new collection with elements with tag equal to tag0 or tag1 and so on. |
| set\_elem\_model(elem\_model) | None | Sets the analysis model for all elements in the collection. |

## Class <Face>:

### Properties:

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **In/Out** | **Description** |
| id | int | read-only | Represents the face Id based on the faces list of the container Domain object. |
| tag | str | read-write | Label for the face |
| nodes | CollectionNode | read-only | Collection with all face nodes |
| max\_x | float | read only | Maximum x coordinate in collection. |
| max\_y | float | read only | Maximum y coordinate in collection. |
| max\_z | float | read only | Maximum z coordinate in collection. |
| min\_x | float | read only | Minimum x coordinate in collection. |
| min\_y | float | read only | Minimum y coordinate in collection. |
| min\_z | float | read only | Minimum z coordinate in collection. |

**Methods:**

|  |  |  |
| --- | --- | --- |
| **Method** | **Returns** | **Description** |
| set\_brys(brys) | None | Sets the Face object boundary conditions. The parameter brys should be a dict object containing the respective boundary conditions keys and boundary conditions values. Face boundary conditions can be of type Newmann or Dirichlet boundary conditions. |

## Class <CollectionFace>:

Represents a collection of Face objects.

**Methods:**

|  |  |  |
| --- | --- | --- |
| **Method** | **Returns** | **Description** |
| extend(face\_cltn) | None | Extends current collection with a given CollectionFace face\_cltn. |
| with\_tag(tag) | CollectionFace | Get a new list of faces that match de given tag. |
| with\_x(x1,x2,…) | CollectionFace | Get a new CollectionFace with faces which all x coordinates matches one of the given values x1,x2,… |
| with\_y(y1,y2,…) | CollectionFace | Get a new CollectionFace with faces which all y coordinates matches one of the given values y1,y2,… |
| with\_z(z1,z2,…) | CollectionFace | Get a new CollectionFace with faces which all z coordinates matches one of the given values z1,z2,… |
| With\_x\_in\_interval(x0,x1) | CollectionNode | Filters the CollectionFace object with all nodes with x coordinate inside the closed interval [x0,x1]. |
| With\_y\_in\_interval(y0,y1) | CollectionNode | Analogous to with\_x\_**in**\_interval function. |
| With\_x\_in\_interval(z0,z1) | CollectionNode | Analogous to with\_x\_**in**\_interval function. |

## Class <Domain>:

### Properties:

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **In/Out** | **Description** |
| id | int | read-only | Represents the object Id. |
| nodes | CollectionNode | read-only | Collection with all domain nodes. |
| elems | CollectionElem | read-only | Collection with all domain elements. |
| faces | CollectionFace | read-only | Collection with all domain faces. |
| ips | CollectionIp | read-only | Collection with all elements ips from the domain |
| solver | Solver | read-only | Solver object for the current analysis |

### Methods:

|  |  |  |
| --- | --- | --- |
| **Method** | **Returns** | **Description** |
| load\_mesh(mesh) | None | Sets the Domain object with nodes, elements and faces according to the given mesh object mesh. |
| set\_solver(asolver) | None | Sets a specific solver object for finite elements according to asolver. The Solver object can be one of the following classes: SolverEq, SolverFlow, SolverHydroMec, SolverDyn, etc. |
| set\_thickness(thk) | None | Sets the model thickness thk for 2D analyses. |

### Example:

domain = Domain() *# Creates a Domain object*

domain.load\_mesh(‘my\_file.vtk’) *# Load a mesh* ***from*** *file*

*# domain.nodes represents a collection of all domain’s nodes*

*# top\_nodes represents all nodes with z coordinate freater than 10.0*

top\_nodes = domain.nodes.with\_z(10.0)

top\_nodes.set\_bry(uz = 0.0) *# Set boundary conditions* ***in*** *top\_nodes*

# Equilibrium library (equilib):

## Library Functions:

## Class <EqElasticBar>:

## Class <EqDruckerPrager>:

## Class <ElemEq>:

### Example:

elem = Element() # Creates a new Element object

# Defines element functionality for equilibrium analysis:

elem.set\_base(ElemEq())

# Sets the material model (Linear Elastic) for the element:

elem.set\_material(MatLinElastic(E=2000, nu=0.25))

## Class <SolverEq>:

Examples:

domain = Domain() *# Creates a Domain object*

domain.load\_mesh(“my\_file.vtk”) *# Load a mesh* ***from*** *file*

domain.set\_solver(SolverEq()) *# Sets solver* ***for*** *equilibrium*

*# Apply boundary conditions:*

top\_nodes = domain.nodes.filter(lambda n: n.z==10.0)

top\_nodes.set\_bry(“fz”, -10.0)

base\_nodes = domain.nodes.filter(lambda n: n.z==0.0)

base\_nodes.set\_bry(“uz”, 0.0, “uy”, 0.0, “uz”,0.0)

domain.solver.solve() *# Solves the problem*

# Seepage library (seepage)

# Hydro-mechanical library (hydromec)

# Dynamic library (dynamic)