myfempy

Release latest

Antonio Vinicius Garcia Campos.

Feb 23, 2023

CONTENTS

Under Development

CHAPTER ONE

ABOUT

Myfempy is a python package based on finite element method for scientific analysis. The code is open source and *intended for educational and scientific purposes only, not recommended to commercial use*. You can help us by contributing with a donation on the main project page, read the support options. **If you use myfempy in your research, the developers would be grateful if you could cite in your work.**

CHAPTER TWO

INSTALLATION

2.1 To install myfempy manually in your directory, following the steps

1. Clone/ Download the main code [latest version] from github/myfempy/main
2. Unzip the pack in your preferred location
3. In the **myfempy-main** folder, open a terminal and enter with the command:

```
>> python -m pip install --upgrade pip

>> pip install .

or

>> python -m pip install --upgrade build

>> python -m build
```

Note: is recommend to create a virtual environment previously the installation of myfempy** and dependencies packs. You can use the virtualenv or conda environments**

CHAPTER

THREE

DEPENDENCIES

Myfempy can be used in systems based on Linux, MacOS and Windows. **Myfempy** requires Python 3.

3.1 Installation prerequisites, required to build myfempy

You can use either of two python development environments to run myfempy

- Python 3.x - *Python is a programming language that lets you work quickly and integrate systems more effectively.*
- Anaconda - *Anaconda offers the easiest way to perform Python/R data science and machine learning on a single machine.*

3.2 Python packages required for using myfempy

The following python packages are required to run myfempy. Before to install myfempy-main, install this packages. Check if they are already installed on your machine

- numpy - The fundamental package for scientific computing with Python
- cython - Cython is a language that makes writing C extensions for Python as easy as Python itself
- scipy - Fundamental algorithms for scientific computing in Python
- vedo - A python module for scientific analysis and visualization of d objects
- vtk(optional) - VTK is an open-source toolkit for 3D computer graphics, image processing, and visualization
- try

```
>> pip install numpy, cython, scipy, vedo
```

3.2.1 Outhers prerequisites

- gmsh/External Generator Mesh - Gmsh is an open source 3D finite element mesh generator with a built-in CAD engine and post-processor. *Notes: 1 - Gmsh is NOT part of myfempy projects; 2 - Is Needed install Gmsh manually*
- try

```
>> pip install --upgrade gmsh
```

- gmsh PyPi

CHAPTER

FOUR

TUTORIAL

A **Basic Tutorial** is available here.

Many **Examples** are available here.

CHAPTER FIVE

DOCUMENTATION

The myfempy is documented using Sphinx under **doc**. The myfempy's documents versions can be found in html, pdf or epub.

The **Web Documentation** is available on [Read the Docs](https://myfempy.readthedocs.io/).

To compile the documentation use *sphinx* in the **doc** folder. Do,

```
>> make html {in the root folder where the index.rst file is} --> This command generates␣
↪*.html* files

>> make latexpdf # [optional] todo doc pdf
```

CHAPTER SIX

RELEASE

The all release versions is available here

CHAPTER SEVEN

FEATURES

The *main myfempy features* are available here:

- Features List

CHAPTER EIGHT

LICENSE

myfempy is published under the GPLv3 license. See the myfempy/LICENSE.

CHAPTER

NINE

CITING

Have you found this software useful for your research? Star the project and cite it as:

- APA:

```
Antonio Vinicius Garcia Campos. (2022). myfempy (1.5.1). Zenodo. https://doi.org/10.5281/
↪zenodo.6958796
```

- BibTex:

```
@software{antonio_vinicius_garcia_campos_2022_6958796,
author        = {Antonio Vinicius Garcia Campos},
title         = {myfempy},
month         = aug,
year          = 2022,
publisher     = {Zenodo},
version       = {1.5.1},
doi           = {10.5281/zenodo.6958796},
url           = {https://doi.org/10.5281/zenodo.6958796}
}
```

CHAPTER TEN

REFERENCES

- Myfempy - *A python package for scientific analysis based on finite element method.*
- FEM - *The finite element method (FEM) is a popular method for numerically solving differential equations arising in engineering and mathematical modeling.*
- Solid Mechanics - *Solid mechanics, also known as mechanics of solids, is the branch of continuum mechanics that studies the behavior of solid materials, especially their motion and deformation under the action of forces, temperature changes, phase changes, and other external or internal agents.*
- PDE - *In mathematics, a partial differential equation (PDE) is an equation which imposes relations between the various partial derivatives of a multivariable function.*

CHAPTER ELEVEN

CHANGELOG

The changelog is available here

CHAPTER

TWELVE

PROJECT TREE STRUCTURE

```
/myfempy
|   version.py
|   __init__.py
|
+---core
|       assembler.py
|       solver.py
|       solverset.py
|       staticlinear.py
|       vibralinear.py
|       __init__.py
|
+---felib
|   |   crossec.py
|   |   felemset.py
|   |   materset.py
|   |   physicset.py
|   |   quadrature.py
|   |   __init__.py
|   |
|   +---fluid
|   |       __init__.py
|   |
|   +---fsi
|   |       __init__.py
|   |
|   +---materials
|   |       axial.py
|   |       lumped.py
|   |       planestrain.py
|   |       planestress.py
|   |       solid.py
|   |       __init__.py
|   |
|   +---physics
|   |       force2node.py
|   |       getnode.py
|   |       loadsconstr.py
|   |       __init__.py
|   |
```

(continues on next page)

(continued from previous page)

```
|   \---struct
|           beam21.py
|           frame21.py
|           frame22.py
|           plane31.py
|           plane41.py
|           solid41.py
|           solid81.py
|           spring21.py
|           truss21.py
|           __init__.py
|
+---io
|       filters.py
|       iomsh.py
|       iovtk.py
|       __init__.py
|
+---mesh
|       genmesh.py
|       gmsh.py
|       legacy.py
|       __init__.py
|
+---plots
|       meshquality.py
|       physics.py
|       plotmesh.py
|       plotxy.py
|       postplot.py
|       prevplot.py
|       __init__.py
|
+---postprc
|       displcalc.py
|       postcomp.py
|       postset.py
|       __init__.py
|
\---tools
        logo.png
        logo.txt
        path.py
        tools.py
        __init__.py
```

12.1 Basic Tutorial

12.2 Documentation

12.2.1 Introduction

12.2.2 Installation

12.2.3 User's Guide

Inputs Setting

Pre-Process

`myfempy.mesh.genmesh.ModelGen.get_model(meshdata: dict{})`

Model Setting

`meshdata{"PROPMAT"}: list[mat_set_1: dict{}, ..., mat_set_n: dict{}]`

```
mat_set_n = {
# parameters
    "NAME":str(def.val.='mat_1')             # material name def
    "EXX":float(def.val.=1.0)                # elasticity modulus in x direction␣
↪[link](https://en.wikipedia.org/wiki/Young%27s_modulus)
    "VXX":float(def.val.=1.0)                # poisson's ratio in x direction ␣
↪[link](https://en.wikipedia.org/wiki/Poisson%27s_ratio)
    "GXX":float(def.val.=1.0)                # shear modulus in x direction ␣
↪[link](https://en.wikipedia.org/wiki/Shear_modulus)
    "EYY":float(optional)                    # elasticity modulus in y direction, to␣
↪orthotropic material only
    "VYY":float(optional)                    # poisson's ratio in y direction, to␣
↪orthotropic material only
    "GYY":float(optional)                    # shear modulus in y direction, to␣
↪orthotropic material only
    "RHO":float(optional)                    # density, to dynamic analysis only␣
↪[link](https://en.wikipedia.org/wiki/Density)
    "STIF":float(optional)                   # stiffness lumped, to lumped model
    "DAMP":float(optional)                   # damping lumped, to lumped model
    "MAT":str(def.val.='isotropic')          # material definition
        # options
            'springlinear'                   # spring linear lumped
            'springnonlin'                   # spring non linear lumped
            'isotropic'                      # isotropic stress/strain material
            'orthotropic'                    # orthotropic stress/strain material
    "DEF":str(def.val.='planestress')        # material behavior
        # options
            'lumped'                         # lumped material
            'axial'                          # axial{rod, beams...} behavior material
            'planestress'                    # plane stress behavior
```

(continues on next page)

(continued from previous page)

```
            'planestrain'                   # plane strain behavior
            'solid'                         # solid behavior material
```

meshdata{"PROPGEO"}: list[geo_set_1: dict{}, ..., geo_set_n: dict{}]

```
geo_set_n = {
# parameters
    "NAME":str(def.val.='geo_1')            # geometry name def
    "AREACS":float(def.val.=1.0)            # area cross section
    "INERXX":float(def.val.=1.0)            # inercia x diretion [link](https://en.
↪wikipedia.org/wiki/List_of_moments_of_inertia)
    "INERYY":float(def.val.=1.0)            # inercia y diretion
    "INERZZ":float(def.val.=1.0)            # inercia z diretion
    "THICKN":float(def.val.=1.0)            # thickness of plane/plate
    "SEC":str(optional)                     # type of cross section, view list
    "DIM":dict(optional)(def.val.={         # dimensional cross section def, view list
        "b":float(def.val.=1.0)             # b size
        "h":float(def.val.=1.0)             # h size
        "t":float(def.val.=1.0)             # t size
        "d":float(def.val.=1.0)})           # d size
```

meshdata{"FORCES"}: list[force_set_1: dict{},..., force_set_n: dict{}]

```
force_set_n = {
# parameters
    "DEF":str(def.val.='forcenode')         # type force n def.
        # options
            'forcenode'                     # force in nodes, concentrated load
            'forceedge'                     # force in edge, distributed load
            'forcebeam'                     # force in beam only opt., distributed load␣
↪[legacy version]
            'forcesurf'                     # force in surface, distributed load
    "DOF":str(def.val.='fx')                # dof direction of force n
        # options
            'fx'                            # force in x dir.
            'fy'                            # force in y dir.
            'fz'                            # force in z dir.
            'tx'                            # torque/moment in x dir.
            'ty'                            # torque/moment in y dir.
            'tz'                            # torque/moment in z dir.
            'masspoint'                     # mass concentrated applied in node/point
            'spring2ground'                 # spring connected node to ground/fixed end
            'damper2ground'                 # damper connected node to ground/fixed end
    "DIR":str(def.val.='node')              # type direction of force n
        # options
            # ----- OPT. WITH LOC SEEKERS
            'node'                          # node in mesh
            'lengthx'                       # length line in x dir., beam only option␣
↪[legacy version]
            'lengthy'                       # length line in y dir., beam only option␣
↪[legacy version]
            'lengthz'                       # length line in z dir., beam only option␣
```

(continues on next page)

(continued from previous page)

```
↪[legacy version]
            'edgex'                         # edge def in x dir. >'LOC': {'x':float(coord.␣
↪x nodes), 'y':999(select all node in y dir.), 'z':float(coord. z nodes)}
            'edgey'                         # edge def in y dir.
            'edgez'                         # edge def in z dir.
            'surfxy'                        # surf def in xy plane >'LOC': {'x':999, 'y':␣
↪999, 'z':float(coord. z nodes)}
            'surfyz'                        # surf def in yz plane
            'surfzx'                        # surf def in zx plane
            # ----- OPT. WITH TAG SEEKERS
            'point'                         # point number in tag list
            'edge'                          # edge number in tag list
            'surf'                          # surface number in tag list
    "LOC":dict(def.val.={                   # coord. node locator of force n
        'x':float(def.val.=1.0)             # x coord. node
        'y':float(def.val.=1.0)             # y coord. node
        'z':float(def.val.=0.0)})           # z coord. node
    "TAG":int(optional)                     # tag number of regions type, used with gmsh␣
↪mesh gen, view list
    "VAL":list(def.val.=[-1.0])             # value list of force on steps, signal +/-␣
↪is the direction
        # options
            [val_force_step_1,              # force on steps, in solver opt. is possible␣
↪to indicate the one step or all steps number
            ...,
            val_force_step_n]
```

`meshdata{"BOUNDCOND"}: list[boundcond_set_1: dict{},..., boundcond_set_n: dict{}]`

```
boundcond_set_n = {
# parameters
    "DEF":str(def.val.='fixed')             # type force n def.
        # options
            'fixed'                         # fixed boundary condition u=0. More in␣
↪[link](https://en.wikipedia.org/wiki/Boundary_value_problem)
            'displ'                         # displ boundary condition u!=0. [dev]
    "DOF":str(def.val.='all')               # dof direction of force n
        # options
            'ux'                            # force in x dir.
            'uy'                            # force in y dir.
            'uz'                            # force in z dir.
            'rx'                            # torque/moment in x dir.
            'ry'                            # torque/moment in y dir.
            'rz'                            # torque/moment in z dir.
            'all'                           # mass concentrated applied in node/point
    "DIR":str(def.val.='edgex')             # type direction of force n
        # options
            # ----- OPT. WITH LOC SEEKERS
            'node'                          # node in mesh
            'edgex'                         # edge def in x dir. >'LOC': {'x':float(coord.␣
↪x nodes), 'y':999(select all node in y dir.), 'z':float(coord. z nodes)}
            'edgey'                         # edge def in y dir.
```

(continues on next page)

(continued from previous page)

```
            'edgez'                     # edge def in z dir.
            'surfxy'                    # surf def in xy plane >'LOC': {'x':999, 'y':␣
↪999, 'z':float(coord. z nodes)}
            'surfyz'                    # surf def in yz plane
            'surfzx'                    # surf def in zx plane
            # ----- OPT. WITH TAG SEEKERS
            'point'                     # point number in tag list
            'edge'                      # edge number in tag list
            'surf'                      # surface number in tag list
    "LOC":dict(def.val.={               # coord. node locator of force n
        'x':float(def.val.=0.0)         # x coord. node
        'y':float(def.val.=999)         # y coord. node
        'z':float(def.val.=0.0)})       # z coord. node
    "TAG":int(optional)                 # tag number of regions type, used with gmsh␣
↪mesh gen, view list
    "VAL":list(def.val.=[1.0])          # value list of dislp on steps [dev]
        # options
            [val_displ_step_1,          # dislp on steps, in solver opt. is possible␣
↪to indicate the one step or all steps number
            ...,
            val_displ_step_n]
```

See Table 3 Consistent Units

`meshdata{"QUADRATURE"}: dict{}`

```
# parameters
    'meth':str(def.val.='no_interpol')      # method to integration
        # options
            'gaussian'                      # [link](https://en.wikipedia.org/wiki/
↪Gaussian_quadrature)
            'no_interpol'
    'npp':int(def.val.=0)                   # number of points to integrations
        # options
            1
            2
            3
            4
            8
```

`meshdata{"DOMAIN"}: str`

```
# options
    'structural'                        # set a structural model
```

Mesh Legacy options

`meshdata{"LEGACY"}: dict{} # LEGACY mesh return a rectangular plane only [test option]`

```
# parameters
    'lx':float(def.val.=1.0)                # set a length in x diretion
    'ly':float(def.val.=1.0)                # set a length in y diretion
    'nx':int(def.val.=10)                   # set a number of elements in x diretion
    'yx':int(def.val.=10)                   # set a number of elements in y diretion
    'mesh':str(def.val.=tria3)              # set a type of mesh used in analysis
        <goto> Table 1 Mesh List
    'elem':str(def.val.=plane31)            # set a type of element used in analysis
        <goto> Table 2 Elements List
```

`meshdata{"ELEMLIST"}: list[] # ELEMLIST return a element list from a manual mesh [old option]`

```
# set
    [
    [elem_number_n:int, 'elem':str, mat_set_n{'NAME'}(set first mat_set_n:dict{}), geo_
↪set_n{'NAME'}(set first geo_set_n:dict{}), nodes_list_conec_n:list[]]
    ...
    ]
    >> [[1, 'plane31', 'steel', 'geo', [1, 2, 3]]]
```

`meshdata{"NODELIST"}: list[] # NODELIST return a nodes list from a manual mesh [old option]`

```
# set
    [
    [node_number_n:int, coord_x:float, coord_y:float, coord_z:float]
    ...
    ]
    >> [[1, 0, 0, 0]
        [2, 1, 0, 0]
        [3, 0, 1, 0]]
```

Gmsh Mesh options

Notes: 1 - Gmsh is NOT part of myfempy projects; 2 - Is Needed install Gmsh manually

`meshdata{"GMSH"}: dict{} # GMSH mesh return a advacend mesh from gmsh external lib [link](https://pypi.org/project/gmsh/) [advanced option]`

```
# parameters
    'filename':str                          # name of files exit
    'meshimport':dict{}                     # opt. to import a external gmsh mesh
        # option
            'object':str(object name .msh1) # file .msh1 only, legacy mesh from gmsh␣
↪[current version]
    'cadimport':dict{}                      # opt. to import a cad model from any cad␣
↪program [link](https://en.wikipedia.org/wiki/Computer-aided_design) [FreeCAD](https://
```

(continues on next page)

(continued from previous page)

```
↪www.freecad.org/index.php?lang=pt_BR)
        # option
            'object':str(object name .step) # file .step/.stp only [current version]
    *** Options to build a self model in .geo file (from gmsh)
    'pointlist':list[]                      # poinst coord. list
        # set
            [
            [coord_x_point_1:float, coord_y_point_1:float, coord_z_point_1:float]
            ...
            [coord_x_point_n:float, coord_y_point_n:float, coord_z_point_n:float]
            ]

        #  y
        #  |
        #  |
        # (1)----x
        #   \
        #    \
        #     z

        #-- lines points conec., counterclockwise count
        # set
            [
            [point_i_line_1:int, point_j_line_1:int]
            ...
            [point_i_line_n:int, point_j_line_n:int]
            ]

        # (i)-----{1}-----(j)

    'planelist':list[]                      # planes lines conec., counterclockwise count
        # set
            [
            [line_1_plane_1:int, ..., line_n_plane_1:int]
            ...
            [line_1_plane_n:int, ..., line_n_plane_n:int]
            ]

        # (l)-----{3}-----(k)
        #  |               |
        #  |               |
        # {4}     [1]     {2}
        #  |               |
        #  |               |
        # (i)-----{1}-----(j)

    'arc':list[]                            # arc line set, counterclockwise count
        # set
            [
            [R,[CX,CY,CZ],[A0, A1]] # arc_1
            ...
            [R,[CX,CY,CZ],[A0, A1]] # arc_n
```

(continues on next page)

(continued from previous page)

```
            ]

        #       A1    ^
        #       |    /
        #       |   /
        #       |  R
        #       | /
        #       |/
        # (i:CX,CY,CZ)------A0

        # options
            R:float                         # radius
            CX:float                        # point i center x coord.
            CY:float                        # point i center y coord.
            CZ:float                        # point i center z coord.
            A0:str(def.val.='0')            # angle begin rad
            A1:str(def.val.='Pi/2')         # angle end rad

    'meshconfig':dict{}                     # mesh configuration inputs
        # options
            'mesh':str                      # set a type of mesh used in analysis
                <goto> Table 1 Mesh List
            'elem':str                      # set a type of element used in analysis
                <goto> Table 2 Elements List
            'sizeelement':float             # size min. of elements
            'numbernodes':int               # select a number of nodes in line, only to
↪'line2' <goto> Table 1 Mesh List
            'meshmap':dict{}                # gen. a mapped structured mesh
                # option
                    'on':bool               # turn on(true/ false)
                        True
                        False
                    'edge':two opt.         # select edge to map (only in 'on':True)
                        'numbernodes':int   # select a number of nodes in edge
                        'all'/ TAG NUMB:int # select all edge or a specific edge
            'extrude':float                 # extrude dimensional, in z diretion, from a␣
↪xy plane
```

Preview analysis

Solver Set

Post-Process View

Appendix

Table 1 Mesh list

mesh	supported elements
“line2”	“truss21”, “beam21”, “frame21”, “frame22”
“tria3”	“plane31”
“quad4”	“plane41”
“hexa8”	“solid81”
“tetr4”	“solid41”

Table 2 Elements List

element	key/id	description
‘spring21’	110	spring 2D 2-node linear Finite Element
‘truss21’	120	truss 2D 2-node linear Finite Element
‘beam21’	130	beam 1D 2-node linear Finite Element
‘frame21’	140	frame 2D 2-node linear Finite Element
‘frame22’	141	frame 3D 2-node linear Finite Element
‘plane31’	210	triagular Plane 3-node linear Finite Element
‘plane41’	220	quatrangular Isoparametric Plane 4-node Finite Element
‘plate41’	221	quatrangular Isoparametric Plate Mindlin 4-node Finite Element [dev]
‘solid41’	310	tetrahedron Isoparametric Solid 8-node Finite Element
‘solid81’	320	hexahedron Isoparametric Solid 8-node Finite Element

Table 3 Consistent Units

Quantity	SI(m)	SI(mm)
length	m	mm
force	N	N
mass	kg	ton(kg E03)
time	s	s
stress	Pa(N/m^2)	MPa(N/mm^2)
energy	J	mJ(J E-03)
density	kg/m^3	ton/mm^3

Axis Diretions

```
#        |
#        [Y]
#        |      P1 -- principal plane
#        |      P2 -- secondary plane
#        |__edgey__
#       /|         |
#      / |   P1    |
#     /  | surfxy  edgex
#    /  f|         |
#   /  r |_________|____________[X]__
#  | u  /          /
#  |s z/   P2     /
#  | y/  surfzx  edgez
#  | /          /
#  |/__________/
#  /
# [Z]
#/
```

Cross Section Dimensions

```
#                       :
#                       [Y]
#                       :
#  ___       ___________:___________
#   |       |_______    :    _______|
#   |              |   :   |
#   |            -->|   :   |<-------------(t)
#   |              |   :   |
#   |              |   :   |
#   |              |   :   |
#   (h)            |  (CG).|..........[Z]..
#   |              |       |
#   |              |       |
#   |              |       |
#   |              |       |
#   |        _______|       |_______    ___
#  _|_      |_______________________|   _|_(d)


#           |-----------(b)---------|
```

Tag Legends

- [advanced option]: Inputs advanced options, require a external package
- [current version]: Inputs options in the latest stable version of myfempy
- [dev]: Inputs options in development (next update), to test only
- [legacy version]: Inputs of legacy/old version

12.2.4 Examples

12.2.5 Theory Basic

12.2.6 myfempy

myfempy package

Submodules

myfempy.core package

Submodules

myfempy.core.assembler module

Assembly matrix

`class myfempy.core.assembler.`**`Assembler`**

Bases: `object`

`static assembler`(*modelinfo: dict*, *key: str*)

class assembly matrix

Args:
modelinfo:dict: – F.E. model dict with full information needed key:str – key type of assembly

Returns:
matrix:np.ndarray – assembly matrix

`static loads`(*modelinfo: dict*, *KG: ndarray*)

summary

Args:
modelinfo:dict – F.E. model dict with full information needed KG:np.ndarray – stiffness matrix

Returns:
forcevec – forces vector KG – stiffness matrix updated

myfempy.core.solver module

Solver Manager

`class myfempy.core.solver.Solver`

Bases: `object`

class solver

SLD – scipy sparse linear solver SLI – scipy sparse biconjugate gradient stabilized iteration solver SLIPRE – scipy generalized minimal residual iteration solver EIG – scipy eigenvalues and eigenvectors solver FRF – scipy sparse linear steps(frequency) solver

`static get_modal_solve`(*solverset: dict*, *modelinfo: dict*)

get a modal solution

Arguments:
solverset:dict – solver setting modelinfo:dict – F.E. model dict with full information needed

Returns:
solution:dict – solution

`static get_static_solve`(*solverset: dict*, *modelinfo: dict*)

get a static solution

Arguments:
solverset:dict – solver setting modelinfo:dict – F.E. model dict with full information needed

Returns:
solution:dict – solution

myfempy.core.solverset module

Solver Setting

`myfempy.core.solverset.get_constrains_dofs`(*modelinfo: dict*)

get constrains dofs in model

Arguments:
modelinfo:dict – F.E. model dict with full information needed

Returns:
freedof:np.ndarray – free dofs vector fixedof:np.ndarray – fixed dofs vector

`myfempy.core.solverset.get_solve`(*solver_type: str*)

get solver type

SLD – scipy sparse linear solver SLI – scipy sparse biconjugate gradient stabilized iteration solver SLIPRE – scipy generalized minimal residual iteration solver EIG – scipy eigenvalues and eigenvectors solver FRF – scipy sparse linear steps(frequency) solver

`myfempy.core.solverset.step_setting`(*steps: dict*)

steps setting

myfempy.core.staticlinear module

Static Linear Solver

`myfempy.core.staticlinear.`**`sld`**(*fulldofs: int*, *stiffness: ndarray*, *forcelist: ndarray*, *freedof: ndarray*, *solverset: dict*)

scipy sparse linear solver

`myfempy.core.staticlinear.`**`sli`**(*fulldofs: int*, *stiffness: ndarray*, *forcelist: ndarray*, *freedof: ndarray*, *solverset: dict*)

scipy sparse biconjugate gradient stabilized iteration solver

`myfempy.core.staticlinear.`**`slipre`**(*fulldofs: int*, *stiffness: ndarray*, *forcelist: ndarray*, *freedof: ndarray*, *solverset: dict*)

scipy generalized minimal residual iteration solver

myfempy.core.vibralinear module

Vibration/Dynamic Linear Solver

`myfempy.core.vibralinear.`**`eig`**(*fulldofs: int*, *stiffness: ndarray*, *mass: ndarray*, *forcelist: ndarray*, *freedof: ndarray*, *solverset: dict*)

scipy eigenvalues and eigenvectors solver

`myfempy.core.vibralinear.`**`frf`**(*fulldofs: int*, *stiffness: ndarray*, *mass: ndarray*, *forcelist: ndarray*, *freedof: ndarray*, *solverset: dict*)

scipy sparse linear steps(frequency) solver

Module contents

myfempy.felib package

Subpackages

myfempy.felib.fluid package

Module contents

myfempy.felib.fsi package

Module contents

myfempy.felib.materials package

Submodules

myfempy.felib.materials.axial module

axial.py: Axial Isotropic material

class `myfempy.felib.materials.axial.`**`Elasticity`**(*tabmat: ndarray*, *inci: ndarray*, *num_elm: int*)

Bases: `object`

elasticity set class

`isotropic`()

isotropic def

Returns:
D:list[] – elasticity matrix

class `myfempy.felib.materials.axial.`**`Tensor`**(*modelinfo: dict*, *U: ndarray*, *ee: int*)

Bases: `object`

material tensor stress-strain relat.

`strain`()

strain in element

Returns:
epsilon:list[] – list of strain calc. [e] = [B]*{U} strain:float – list of strain tensor title:list[] – tensor set names myfempy

`stress`(*epsilon: ndarray*)

stress in element

Arguments:
epsilon:np.array[] – strain in element

Returns:
stress:list[] – list of stress calc. [s] = [D]*[e] title:list[] – tensor set names myfempy

myfempy.felib.materials.lumped module

myfempy.felib.materials.planestrain module

planestrain.py: Plane Strain Isotropic material

class `myfempy.felib.materials.planestrain.`**`Elasticity`**(*tabmat: ndarray*, *inci: ndarray*, *num_elm: int*)

Bases: `object`

elasticity set class

`isotropic`()

_sotropic def

Returns:
D:list[] – elasticity matrix

myfempy.felib.materials.planestress module

planestress.py: Plane Stress Isotropic and Elasticity Material

class `myfempy.felib.materials.planestress.`**`Elasticity`**(*tabmat: ndarray*, *inci: ndarray*, *num_elm: int*)

Bases: `object`

elasticity set class

`isotropic`()

isotropic def

Returns:
D:list[] – elasticity matrix

class `myfempy.felib.materials.planestress.`**`Tensor`**(*modelinfo: dict*, *U: ndarray*, *ee: int*)

Bases: `object`

_material tensor stress-strain relat.

`strain`()

strain in element

Returns:
epsilon:list[] – list of strain calc. [e] = [B]*{U} strain:float – list of strain tensor title:list[] – tensor set names myfempy

`stress`(*epsilon: ndarray*)

stress in element

Arguments:
epsilon:np.array[] – strain in element

Returns:
stress:list[] – list of stress calc. [s] = [D]*[e] title:list[] – tensor set names myfempy

myfempy.felib.materials.plate module

planestress.py: Plane Stress Isotropic and Elasticity Material

class `myfempy.felib.materials.plate.`**`Elasticity`**(*tabmat: ndarray*, *inci: ndarray*, *num_elm: int*)

Bases: `object`

summary

`isotropic`()

summary

Returns:
description

class `myfempy.felib.materials.plate.`**`Tensor`**(*modelinfo: dict*, *U: ndarray*, *ee: int*)

Bases: `object`

summary

`strain`()

summary

Returns:
description

`stress`(*epsilon: ndarray*)

summary

Arguments:
epsilon – _description_

Returns:
description

myfempy.felib.materials.solid module

solid.py: Solid Isotropic and Elasticity Material

`class myfempy.felib.materials.solid.Elasticity`(*tabmat: ndarray*, *inci: ndarray*, *num_elm: int*)

Bases: `object`

elasticity set class

`isotropic`()

isotropic def

Returns:
D:list[] – elasticity matrix

`class myfempy.felib.materials.solid.Tensor`(*modelinfo: dict*, *U: ndarray*, *ee: int*)

Bases: `object`

_material tensor stress-strain relat.

`strain`()

strain in element

Returns:
epsilon:list[] – list of strain calc. [e] = [B]*{U} strain:float – list of strain tensor title:list[] – tensor set names myfempy

`stress`(*epsilon: ndarray*)

summary

Arguments:
epsilon:np.array[] – strain in element

Returns:
stress:list[] – list of stress calc. [s] = [D]*[e] title:list[] – tensor set names myfempy

Module contents

myfempy.felib.physics package

Submodules

myfempy.felib.physics.force2node module

forces list to nodes vector

`myfempy.felib.physics.force2node.`**`force_beam`**`(`*modelinfo: dict*, *force_value: float*, *force_dirc: str*, *fc_set: str*, *node_list_fc: ndarray*`)`

force in line beam appl.

Arguments:
modelinfo:dict – F.E. model dict with full information needed force_value:float – force value force_dirc:str – force direction node_list_fc:list – list of node with force applied fc_set:str – force set direction

Returns:
force_value_vector:np.array – force vecto fc_type_dof:list – force list dofs

`myfempy.felib.physics.force2node.`**`force_edge`**`(`*modelinfo: dict*, *force_value: float*, *force_dirc: str*, *node_list_fc: ndarray*, *fc_set: str*`)`

force in edge appl.

Arguments:
modelinfo:dict – F.E. model dict with full information needed force_value:float – force value force_dirc:str – force direction node_list_fc:list – list of node with force applied fc_set:str – force set direction

Returns:
force_value_vector:np.array – force vecto fc_type_dof:list – force list dofs

`myfempy.felib.physics.force2node.`**`force_surf`**`(`*modelinfo: dict*, *force_value: float*, *force_dirc: str*, *node_list_fc: ndarray*, *fc_set: str*`)`

force in surface appl.

Arguments:
modelinfo:dict – F.E. model dict with full information needed force_value:float – force value force_dirc:str – force direction node_list_fc:list – list of node with force applied fc_set:str – force set direction

Returns:
force_value_vector:np.array – force vecto fc_type_dof:list – force list dofs

`myfempy.felib.physics.force2node.`**`poly_area`**`(`*poly*`)`

`myfempy.felib.physics.force2node.`**`unit_normal`**`(`*a*, *b*, *c*`)`

myfempy.felib.physics.getnode module

get nodes

`myfempy.felib.physics.getnode.`**`nodes_from_regions`**`(`*regionlist: dict*`)`

nodes from regions tag list

Arguments:
regionlist:dict – regions from tag list (gmsh mesh only)

Returns:
regions:dict

`myfempy.felib.physics.getnode.`**`search_edgex`**`(`*edge_coordX: float*, *coord: ndarray*, *erro: float*`)`

serch. node on x dir. edge

Arguments:
edge_coordX:float – number coord in x dir. coord :np.array – nodes coordinates list in mesh erro:float – erro to conver.

Returns:
node – node loc.

`myfempy.felib.physics.getnode.`**`search_edgey`**`(`*edge_coordY: float*, *coord: ndarray*, *erro: float*`)`

serch. node on y dir. edge

Arguments:
edge_coordY:float – number coord in y dir. coord :np.array – nodes coordinates list in mesh erro:float – erro to conver.

Returns:
node – node loc.

`myfempy.felib.physics.getnode.`**`search_edgez`**`(`*edge_coordZ: float*, *coord: ndarray*, *erro: float*`)`

serch. node on z dir. edge

Arguments:
edge_coordY:float – number coord in z dir. coord :np.array – nodes coordinates list in mesh erro:float – erro to conver.

Returns:
node – node loc.

`myfempy.felib.physics.getnode.`**`search_nodexyz`**`(`*node_coordX: float*, *node_coordY: float*, *node_coordZ: float*, *coord: ndarray*, *erro: float*`)`

serch. node on coord mesh

Arguments:
node_coordX:float – number coord in x dir. node_coordY:float – number coord in y dir. node_coordZ:float – number coord in z dir. coord :np.array – nodes coordinates list in mesh erro:float – erro to conver.

Returns:
node – node loc.

`myfempy.felib.physics.getnode.`**`search_surfxy`**`(`*orthg_coordZ: float*, *coord: ndarray*, *erro: float*`)`

serch. node on z dir. surf

Arguments:
orthg_coordZ:float – number coord in z dir. coord :np.array – nodes coordinates list in mesh erro:float – erro to conver.

Returns:
node – node loc.

`myfempy.felib.physics.getnode.`**`search_surfyz`**`(`*orthg_coordX: float*, *coord: ndarray*, *erro: float*`)`

serch. node on x dir. surf

Arguments:
orthg_coordX:float – number coord in x dir. coord :np.array – nodes coordinates list in mesh erro:float – erro to conver.

Returns:
node – node loc.

`myfempy.felib.physics.getnode.`**`search_surfzx`**`(`*orthg_coordY: float*, *coord: ndarray*, *erro: float*`)`

serch. node ony dir. surf

Arguments:
orthg_coordY:float – number coord in y dir. coord :np.array – nodes coordinates list in mesh erro:float – erro to conver.

Returns:
node – node loc.

myfempy.felib.physics.loadsconstr module

calculate loads and constrains

`myfempy.felib.physics.loadsconstr.`**`get_constrain`**`(`*modelinfo: dict*, *blist: ndarray*`)`

  get bound. cond.

`myfempy.felib.physics.loadsconstr.`**`get_forces`**`(`*modelinfo: dict*, *flist: ndarray*`)`

  get forces

Module contents

myfempy.felib.struct package

Submodules

myfempy.felib.struct.beam21 module

beam21.py: Beam 1D 2-node linear Finite Element

`class` `myfempy.felib.struct.beam21.`**`Beam21`**`(`*modelinfo*`)`

  Bases: `object`

  class Beam 1D 2-node linear Finite Element

  `static elemset``()`

    element setting

  `intforces``(`*U*, *lines*`)`

    internal forces balance calc.

  `lockey``(`*list_node*`)`

    element lockey(dof)

  `mass``(`*ee*`)`

    consistent mass matrix

  `matrix_B``(`*ee*, *csc*`)`

    shape function derivatives

    csc:list[y,z,r] – cross section center(CG)

      y(max,min) – y coord. z(max,min) – z coord. r(max,min) – r(radius) coord.

  `stiff_linear``(`*ee*`)`

    stiffness linear matrix

  `tabgeo`

    Arguments:

      modelinfo:dict – F.E. model dict with full information needed

    Parameters:

      dofe – element dof fulldof – total dof of model nodedof – node dof nelem – total number of elements in mesh nnode – number of degree of freedom per node inci – elements conection and prop. list coord – nodes coordinates list in mesh tabmat – table of material prop. tabgeo – table of geometry prop.

myfempy.felib.struct.frame21 module

frame21.py: Frame 2D 2-node linear Finite Element

class `myfempy.felib.struct.frame21.`**`Frame21`**(*modelinfo*)

Bases: `object`

class Frame 2D 2-node linear Finite Element

static elemset()

element setting

intforces(*U*, *lines*)

internal forces balance calc.

lockey(*list_node*)

element lockey(dof)

mass(*ee*)

consistent mass matrix

matrix_b(*ee*, *csc*)

shape function derivatives

csc:list[y,z,r] – cross section center(CG)
y(max,min) – y coord. z(max,min) – z coord. r(max,min) – r(radius) coord.

stiff_linear(*ee*)

stiffness linear matrix

tabgeo

Arguments:
modelinfo:dict – F.E. model dict with full information needed

Parameters:
dofe – element dof fulldof – total dof of model nodedof – node dof nelem – total number of elements in mesh nnode – number of degree of freedom per node inci – elements conection and prop. list coord – nodes coordinates list in mesh tabmat – table of material prop. tabgeo – table of geometry prop.

myfempy.felib.struct.frame22 module

frame22.py: Frame 3D 2-node linear Finite Element

class `myfempy.felib.struct.frame22.`**`Frame22`**(*modelinfo*)

Bases: `object`

class Frame 3D 2-node linear Finite Element

elemset()

element setting

intforces(*U*, *lines*)

internal forces balance calc.

lockey(*list_node*)

element lockey(dof)

`mass`(*ee*)
: consistent mass matrix

`matrix_b`(*ee*, *csc*)
: shape function derivatives

  csc:list[y,z,r] – cross section center(CG)
  : y(max,min) – y coord. z(max,min) – z coord. r(max,min) – r(radius) coord.

`stiff_linear`(*ee*)
: stiffness linear matrix

`tabgeo`

  Arguments:
  : modelinfo:dict – F.E. model dict with full information needed

  Parameters:
  : dofe – element dof fulldof – total dof of model nodedof – node dof nelem – total number of elements in mesh nnode – number of degree of freedom per node inci – elements conection and prop. list coord – nodes coordinates list in mesh tabmat – table of material prop. tabgeo – table of geometry prop.

myfempy.felib.struct.plane31 module

plane31.py: Triagular Plane 3-node linear Finite Element

`class myfempy.felib.struct.plane31.Plane31`(*modelinfo*)

Bases: `object`

class Beam 1D 2-node linear Finite Element

`static elemset`()
: element setting

`lockey`(*nodelist*)
: element lockey(dof)

`mass`(*ee*)
: consistent mass matrix

`matriz_b`(*nodelist*, *intpl*)
: shape function derivatives

`ntensor`

  Arguments:
  : modelinfo:dict – F.E. model dict with full information needed

  Parameters:
  : dofe – element dof fulldof – total dof of model nodedof – node dof nelem – total number of elements in mesh nnode – number of degree of freedom per node inci – elements conection and prop. list coord – nodes coordinates list in mesh tabmat – table of material prop. tabgeo – table of geometry prop. ntensor – dim. of tensor (stress-strain relat.)

`stiff_linear`(*ee*)
: stiffness linear matrix

myfempy.felib.struct.plane41 module

plane41.py: Quatrangular Isoparametric Plane 4-node linear Finite Element

class `myfempy.felib.struct.plane41.`**`Plane41`**(*modelinfo*)

Bases: `object`

class Quatrangular Isoparametric Plane 4-node linear Finite Element

static **`elemset`**()

element setting

`lockey`(*nodelist*)

element lockey(dof)

`mass`(*ee*)

consistent mass matrix

`matriz_b`(*nodelist*, *intpl*)

shape function derivatives

`stiff_linear`(*ee*)

stiffness linear matrix

myfempy.felib.struct.plate41 module

plate41.py: Quatrangular Isoparametric Plate Mindlin 4-node linear Finite Element

class `myfempy.felib.struct.plate41.`**`Plate41`**(*modelinfo*)

Bases: `object`

class Quatrangular Isoparametric Plate Mindlin 4-node linear Finite Element

static **`elemset`**()

element setting

`lockey`(*nodelist*)

element lockey(dof)

`mass`(*ee*)

consistent mass matrix

`matriz_b`(*nodelist*, *intpl*)

shape function derivatives

`stiff_linear`(*ee*)

stiffness linear matrix

myfempy.felib.struct.solid41 module

solid41.py: Tetrahedron Isoparametric Solid 8-node linear Finite Element

class `myfempy.felib.struct.solid41.`**`Solid41`**(*modelinfo*)

Bases: `object`

class Tetrahedron Isoparametric Solid 8-node linear Finite Element

static **`elemset`**()

element setting

`lockey`(*nodelist*)

element lockey(dof)

`mass`(*ee*)

consistent mass matrix

`matriz_b`(*nodelist*, *intpl*)

shape function derivatives

`stiff_linear`(*ee*)

stiffness linear matrix

myfempy.felib.struct.solid81 module

solid81.py: Hexahedron Isoparametric Solid 8-node linear Finite Element

class `myfempy.felib.struct.solid81.`**`Solid81`**(*modelinfo*)

Bases: `object`

class Hexahedron Isoparametric Solid 8-node linear Finite Element

static **`elemset`**()

element setting

`lockey`(*nodelist*)

element lockey(dof)

`mass`(*ee*)

consistent mass matrix

`matriz_b`(*nodelist*, *intpl*)

shape function derivatives

`stiff_linear`(*ee*)

stiffness linear matrix

myfempy.felib.struct.spring21 module

spring21.py: Spring 2D 2-node linear Finite Element

`class myfempy.felib.struct.spring21.`**`Spring21`**`(`*modelinfo*`)`

Bases: `object`

class Spring 2D 2-node linear Finite Element

`static elemset()`

element setting

`lockey``(`*list_node*`)`

element lockey(dof)

`mass``(`*ee*`)`

consistent mass matrix

`stiff_linear``(`*ee*`)`

stiffness linear matrix

`tabgeo`

Arguments:
modelinfo:dict – F.E. model dict with full information needed

Parameters:
dofe – element dof fulldof – total dof of model nodedof – node dof nelem – total number of elements in mesh nnode – number of degree of freedom per node inci – elements conection and prop. list coord – nodes coordinates list in mesh tabmat – table of material prop. tabgeo – table of geometry prop.

myfempy.felib.struct.truss21 module

truss21.py: Truss 2D 2-node linear Finite Element

`class myfempy.felib.struct.truss21.`**`Truss21`**`(`*modelinfo*`)`

Bases: `object`

class Truss 2D 2-node linear Finite Element

`static elemset()`

element setting

`lockey``(`*list_node*`)`

element lockey(dof)

`matrix_b``(`*ee*, *csc*`)`

shape function derivatives

csc:list[y,z,r] – cross section center(CG)
y(max,min) – y coord. z(max,min) – z coord. r(max,min) – r(radius) coord.

`stiff_linear``(`*ee*`)`

stiffness linear matrix

`tabgeo`

Arguments:
modelinfo:dict – F.E. model dict with full information needed

Parameters:
dofe – element dof fulldof – total dof of model nodedof – node dof nelem – total number of elements in mesh nnode – number of degree of freedom per node inci – elements conection and prop. list coord – nodes coordinates list in mesh tabmat – table of material prop. tabgeo – table of geometry prop.

Module contents

Submodules

myfempy.felib.crossec module

cross section

`myfempy.felib.crossec.`**`cg_coord`**`(`*tabgeo: ndarray*, *inci: ndarray*, *num_elm: int*`)`

coord cg compute

Arguments:
tabgeo:list[] – table of geometry prop. inci:list[] – elements conection and prop. list num_elm:int – element(in mesh) number

Returns:
CG:np.array – coord of CG

`myfempy.felib.crossec.`**`sec_def`**`(`*keysecdef: str*`)`

cross section def

Arguments:
keysecdef:str – key section def

Returns:
idsecdef:int – id number of cross section

`myfempy.felib.crossec.`**`sect_prop`**`(`*sec_set: str*, *dim_sec: dict*`)`

cross section property

Arguments:
sec_set:str – section setting dim_sec:dict{} – section's dimensions

Returns:
A:float – area Izz:float – inercia zz Iyy:float – inercia yy Jxx:float – inercia xx

myfempy.felib.felemset module

Finite Elements Setting

`myfempy.felib.felemset.`**`get_elemset`**`(`*keyelem: str*`)`

get element setting

Arguments:
keyelem:str – key element(view myfempy User's Manual)

Returns:
element class

myfempy.felib.materset module

Material Setting

`myfempy.felib.materset.`**`get_elasticity`**`(`*tabmat: ndarray*, *inci: ndarray*, *num_elm: int*`)`

get elasticity matrix D

Arguments:
tabmat:list[] – table of material prop. inci:list[] – elements conection and prop. list num_elm:int – element(in mesh) number

Returns:
elasticity class

`myfempy.felib.materset.`**`mat_beh`**`(`*keymatbeh: str*`)`

material behavior

Arguments:
keymatbeh:str – key material behavior

Returns:
idmatbeh:int – id mat. beh.

`myfempy.felib.materset.`**`mat_def`**`(`*keymatdef: str*`)`

material def

Arguments:
keymatdef:str – key material def

Returns:
idmatdef:int – id number of cross section

myfempy.felib.physicset module

Physics Setting

`myfempy.felib.physicset.`**`gen_bound`**`(`*boundcondlist: ndarray*`)`

gen boundary conditions set

Arguments:
boundcondlist:list[] – boundary conditions list

Returns:
blist:list[] – boundary conditions list to myfempy

`myfempy.felib.physicset.`**`gen_force`**`(`*forcelist: ndarray*`)`

gen force set

Arguments:
forcelist:list[] – force list in boundary conditions

Returns:
flist:list[] – force list to myfempy

myfempy.felib.quadrature module

Quadrature

`myfempy.felib.quadrature.`**`gaussian`**`(`*npp: int*`)`

integration gauss

Arguments:
npp – number_of_points

Returns:
xp:np.array – points wp:np.array – weights

`myfempy.felib.quadrature.`**`no_interpol`**`(`*npp: int*`)`

no integration

Arguments:
npp – number_of_points

Returns:
xp:np.array – points wp:np.array – weights

Module contents

myfempy.mesh package

Submodules

myfempy.mesh.genmesh module

`class` `myfempy.mesh.genmesh.`**`MeshGen`**

Bases: `object`

generate mesh

`get_data_mesh``()`

get mesh data

Arguments:
meshdata:dict – data model

`class` `myfempy.mesh.genmesh.`**`MeshSet`**

Bases: `object`

class mesh set

`get_coord``()`

get coord nodes

`get_inci``(`*mat_lib: list*, *geo_lib: list*, *regions: list*`)`

get incidence conection

`get_tabgeo``()`

get geometry table

`get_tabmat``()`

get material table

mesh2elem_key()

mesh to elem. key

class myfempy.mesh.genmesh.**ModelGen**

Bases: `object`

generate the model F.E.

get_model()

get model

Arguments:

meshdata:dict – data model

get_quadra()

get quadrature

myfempy.mesh.gmsh module

GMSH GEN MESH

myfempy.mesh.gmsh.**get_gmsh_geo**(*meshdata: dict*)

myfempy.mesh.gmsh.**get_gmsh_msh**(*meshdata: dict*)

myfempy.mesh.gmsh.**gmsh_key**(*meshtype: str*)

myfempy.mesh.legacy module

LEGACY MESH GEN

myfempy.mesh.legacy.**get_legacy_line2**(*GEOMETRY: dict*)

get a line 2 nodes mesh

(i)—{1}—(j)

myfempy.mesh.legacy.**get_legacy_quad4**(*GEOMETRY: dict*)

get a quadrangular 4 nodes mesh

(l)——————(k)

|

|

{1} |

|

|

(i)——————(j)

myfempy.mesh.legacy.**get_legacy_tria3**(*GEOMETRY: dict*)

get a triagular 3 nodes mesh

(k) | | | | | | | {1} | | | |

(i)———(j)

Module contents

PYTHON MODULE INDEX