myfempy

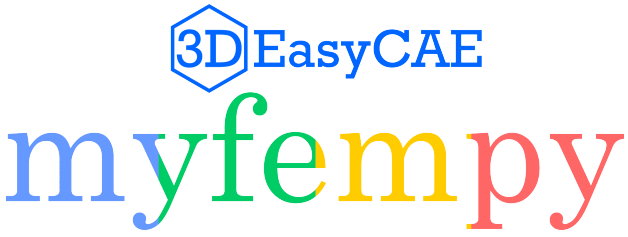
Release latest

Antonio Vinicius Garcia Campos.

Feb 24, 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://
↪www.freecad.org/index.php?lang=pt_BR)
```

(continues on next page)

(continued from previous page)

```
        # 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

myfempy.core.solver module

myfempy.core.solverset module

myfempy.core.staticlinear module

myfempy.core.vibralinear module

Module contents

myfempy.felib package

Subpackages

myfempy.felib.fluid package

Module contents

myfempy.felib.fsi package

Module contents

PYTHON MODULE INDEX

m

myfempy.core, 36

INDEX

M