Introduction

Practical Work (=TP)

Purpose:
1) Understand the main operating principles of a finite element code.
2) Numerically solve mechanical problems of continuous media without analytical solutions: geometry and/or complex boundary conditions.

5 sessions:
December 2: Wings → Introduction, software installation, parameters and representation
December 3: Gravity Dam I → Post-processing and T6 elements
December 9: Gravity Dam II
December 16: Plate with holes I → Boundary conditions, stress concentration
December 17: Plate with holes II

Evaluation on 2 reports:
1) December 15: Gravity Dam
2) January 4: Plate with holes
Reports (introductions/pictures/questions/conclusions) should be in PDF and uploaded before the due date on Moodle.

Documentation and questions in English

Introduction

Code :

- Developped in Python (see « Quick overview of Python language »)
- Low levels (ex: access to local kinematics, to assembly phases,…)
- Written by J. Bleyer (ENPC, IFSTTAR, CNRS UMR 8205)
- Consists mainly of a library called Wombat

Softwares to install :

- **Python + IDE :** programing language and environment.

- **(you can use the Anaconda suite**
- https://www.anaconda.com/download)

ANACONDA

spyder
IDE

qtconsole
GUI

notebook

…

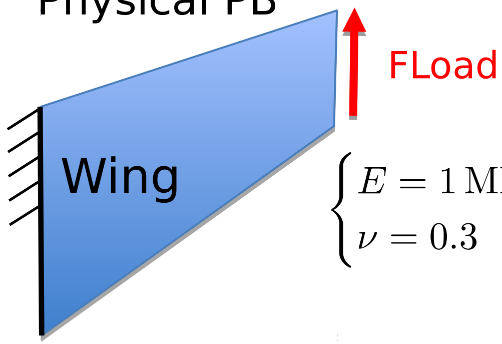
- **The meshio library:** input/output for mesh formats (https://pypi.org/project/meshio/)

- **GMSH** = mesher (open source on Geuzaine's website: http://geuz.org/gmsh/)

gmsh

Sequencing .py files

Physical PB



$$\begin{cases} E = 1\,\mathrm{MPa} \\ \nu = 0.3 \end{cases}$$

Structure of directories and files:

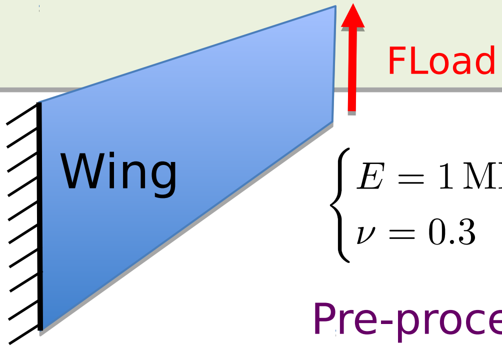
Geometry and mesh

All functions (in Python):
- Boundary conditions
- Elements
- Assembly
- Solving

...

Main (in Python) that:
- Calls the mesh construction
- Uses Wombat functions to solve the FE problem

Sequencing .py files



$$\begin{cases} E = 1\,\mathrm{MPa} \\ \nu = 0.3 \end{cases}$$

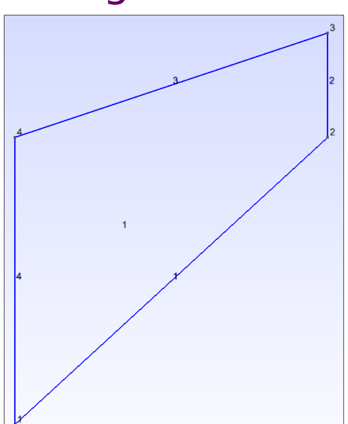
$$\underbrace{\hspace{10em}}_{}$$

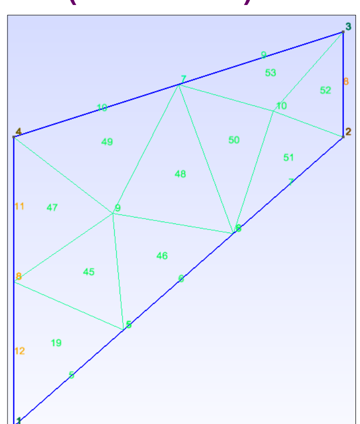
Pre-processing

wing.geo → wing.msh

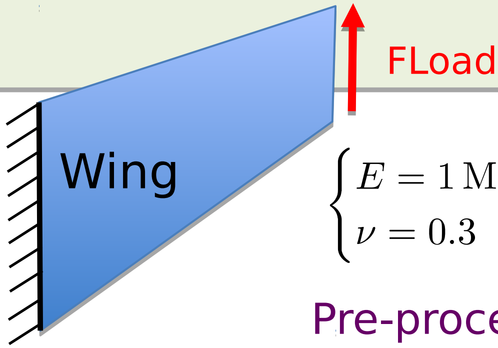
File containing the geometry of the structure
Script to build from Wombat or to generate

File containing the mesh of the structure
Generated through GMSH (mesher)





Sequencing .py files



$$\begin{cases} E = 1\,\mathrm{MPa} \\ \nu = 0.3 \end{cases}$$

Pre-processing

Proper FEA

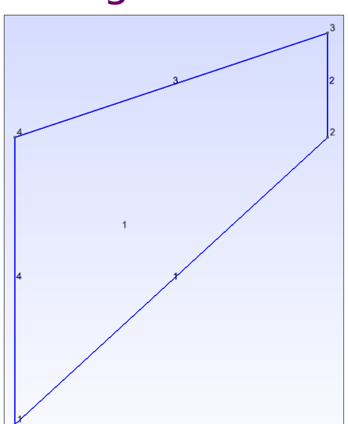
wing.geo → wing.msh → **Wombat Library** → $\{U\} \rightarrow \{\varepsilon\}\,,\ \{\sigma\}$

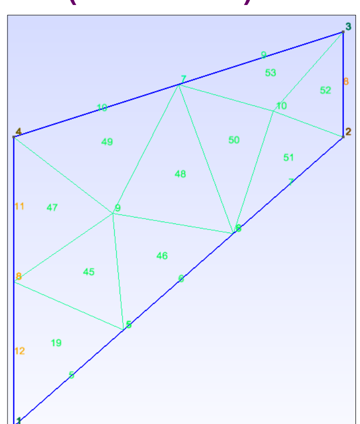
File containing the geometry of the structure
Script to build from Wombat or to generate

File containing the mesh of the structure
Generated through GMSH (mesher)

Wombat Library

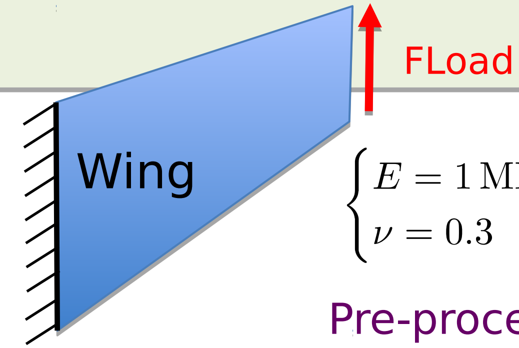
Problem solving
$[K]\,\{U\} = \{F\}$





Physical PB

Sequencing .py files



$$\begin{cases} E = 1\,\mathrm{MPa} \\ \nu = 0.3 \end{cases}$$

Pre-processing

Proper FEA

Post-processing
(integrated)

wing.geo → wing.msh → **Wombat Library**

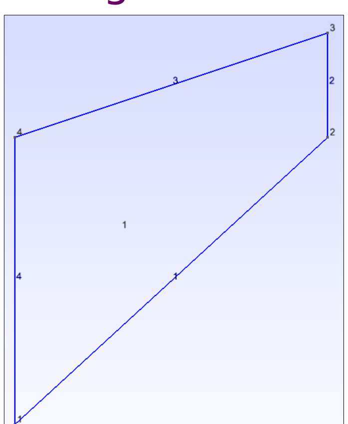
Problem solving
$[K]\{U\} = \{F\}$

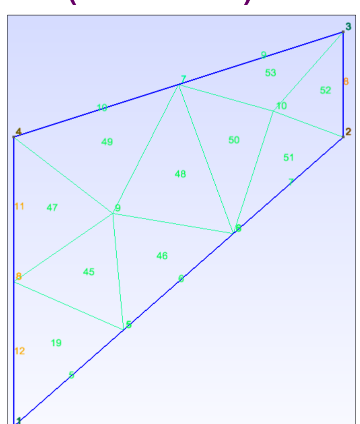
$$\{U\} \to \{\varepsilon\}\,,\ \{\sigma\}$$

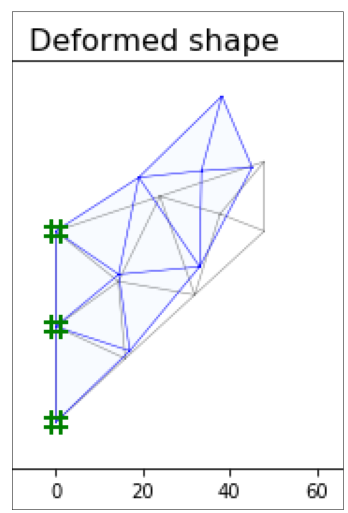
File containing the geometry of the structure
Script to build from Wombat or to generate

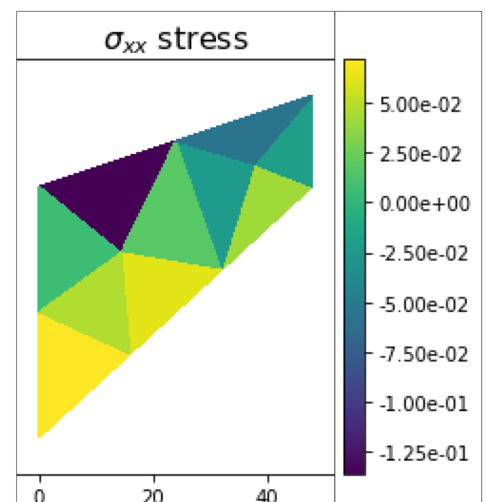
File containing the mesh of the structure
Generated through GMSH (mesher)

Results to visualize (deformed shape, isovalues, etc)

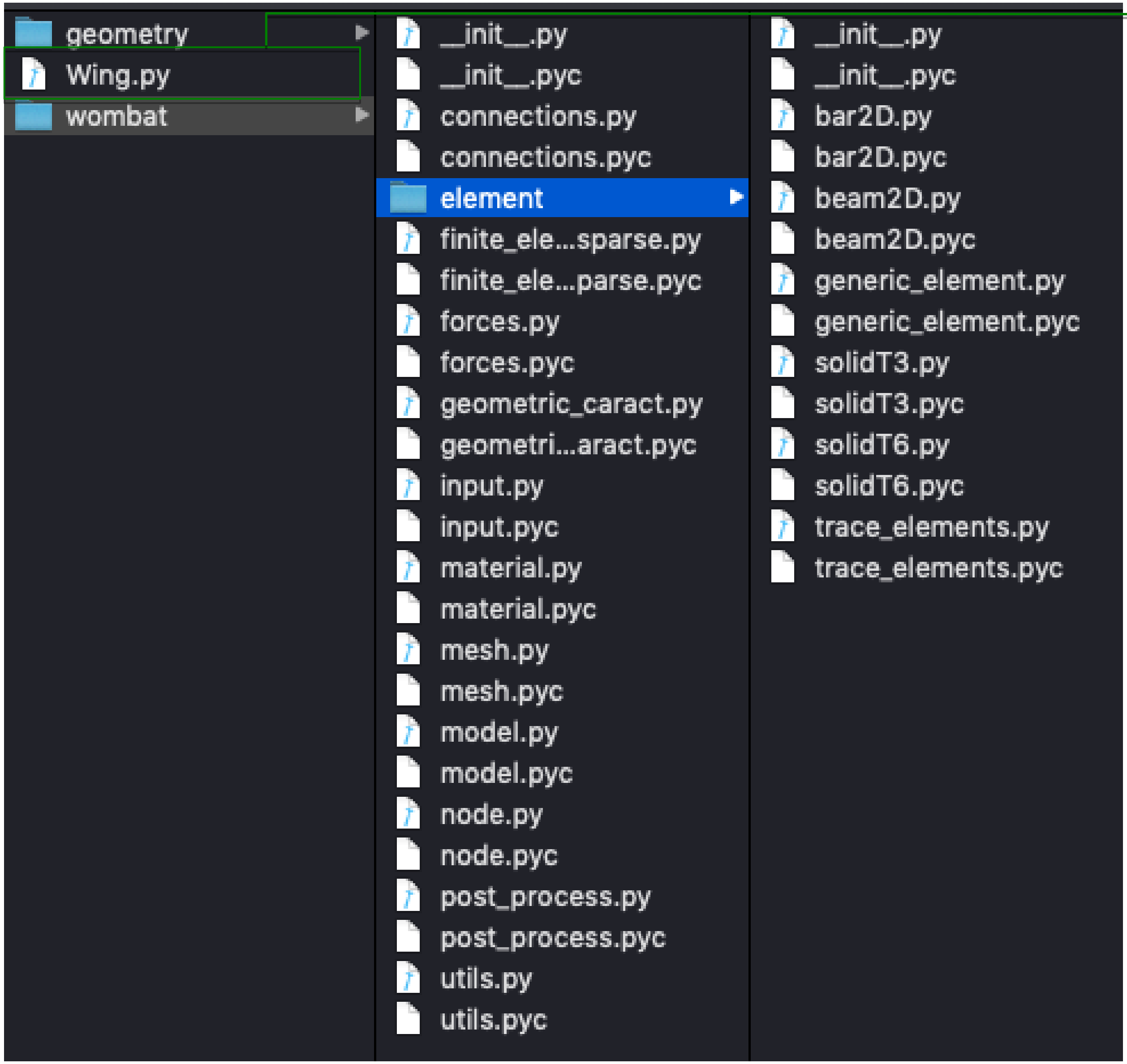








Structure of Wombat



- **Main script**

- Inside **main script** Wing.py:

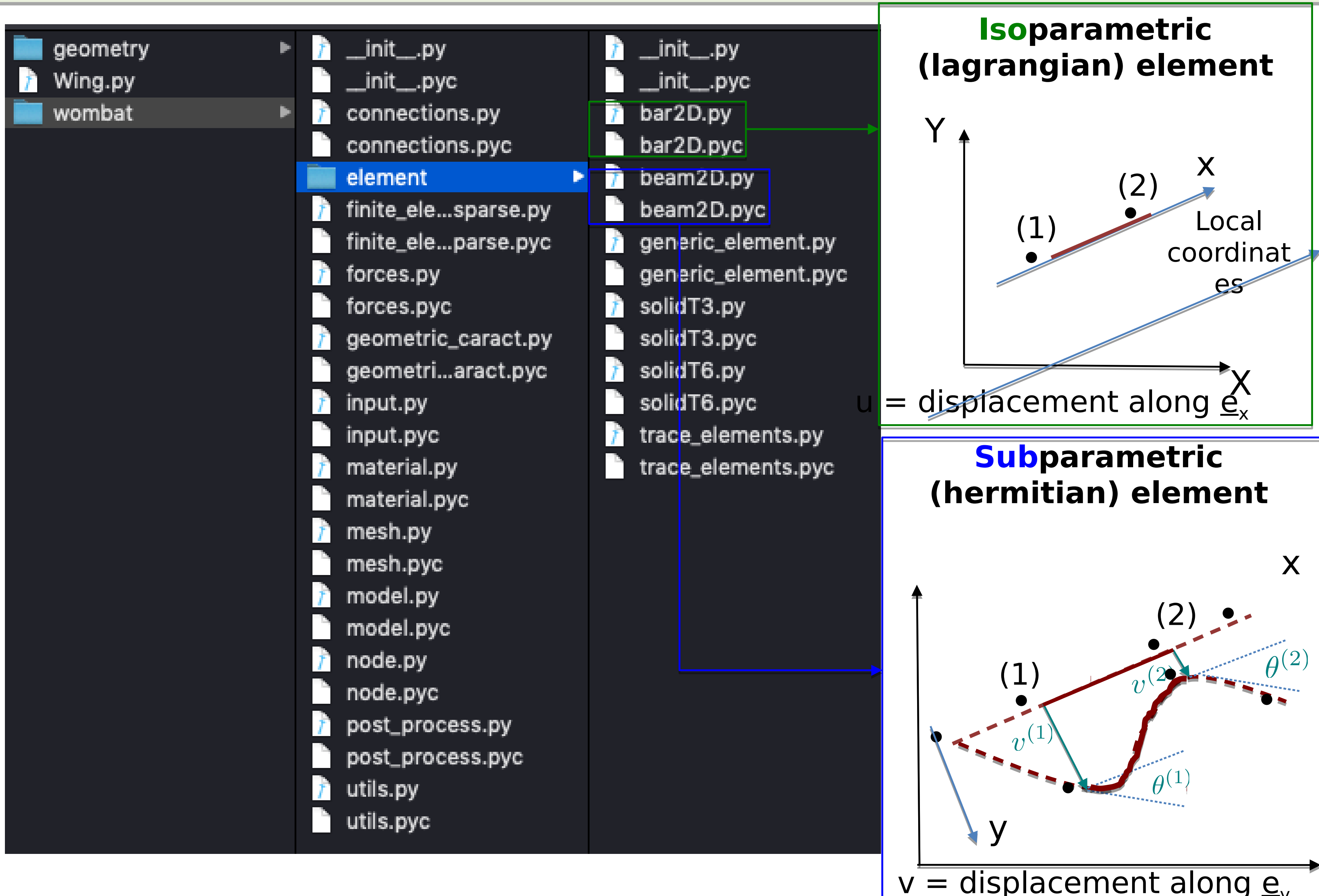
call of classes and methods located inside files of the ***Wombat folder***

- Inside the **element folder**:

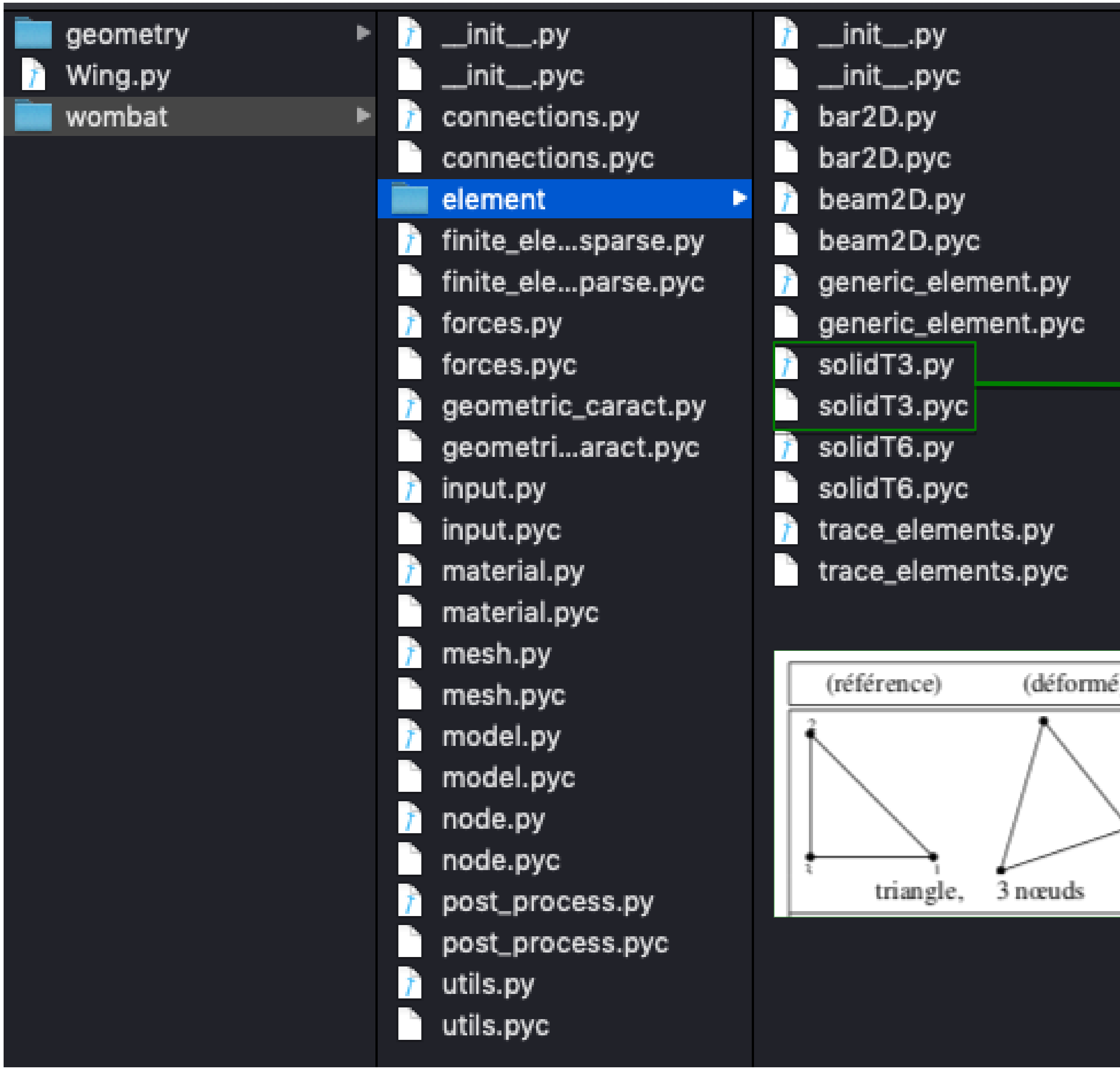
classes and methods concerning the kinematics of elements (shape functions, [Be] & [Ke] matrices, computation of local strains and stresses,…)

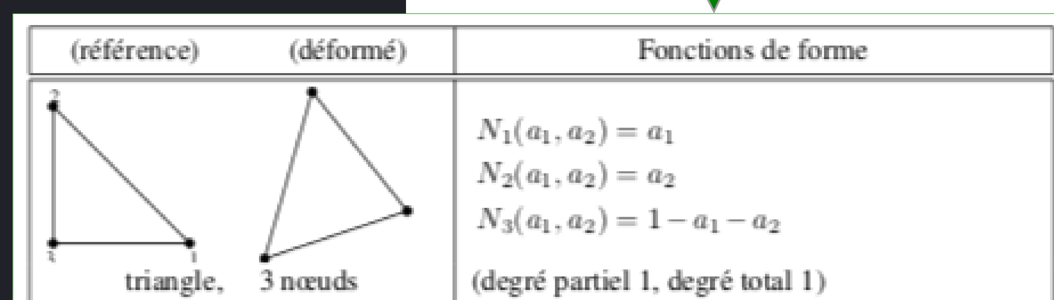
Ex of the code:
to be completed!

Structure of Wombat



Structure of Wombat





(référence) (déformé)	Fonctions de forme
triangle, 3 nœuds	$N_1(a_1, a_2) = a_1$ $N_2(a_1, a_2) = a_2$ $N_3(a_1, a_2) = 1 - a_1 - a_2$ (degré partiel 1, degré total 1)

With: $\begin{cases} a_1 = \xi \text{ (xi)} \\ a_2 = \eta \text{ (eta)} \end{cases}$

Structure of the main script

1. **Import wombat library**
2. **Definition of problem parameters** (size of the structure, values of material properties, of loads…)
3. **Geometry description and mesh generation** (implies prior definition of .geo)
4. **Material properties** (computation of the elasticity tensor,…)
5. **Boundary conditions and load**
6. **Model construction** (assigning materials properties to the corresponding mesh regions,)
7. **Assembly phase** (construction of the stiffness matrix [K] and second member {F}, BC taken into account)
8. **Solving** (of [K]{U}={F})
9. **Post-processing**
   1. Displacements and deformation
   2. Strains and Stresses fields

Always the same steps for any FEA, whatever the code used!

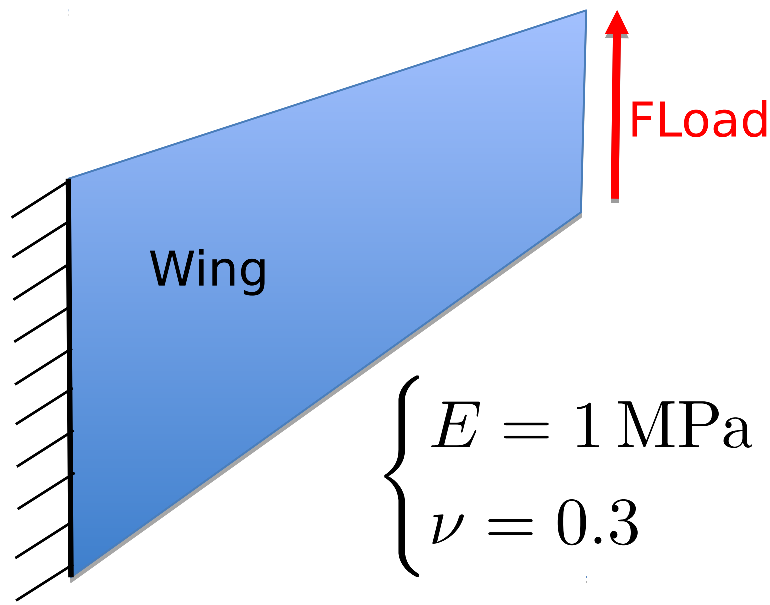
Structure of the main script

1. **Import wombat library**

```python
from wombat import *
```

2. **Definition of problem parameters** (size of the structure, values of material properties, of loads...)

```python
# Problem parameters
Yg_mod = 1
Nu_poi = 0.3
FLoad = 0.0625
```



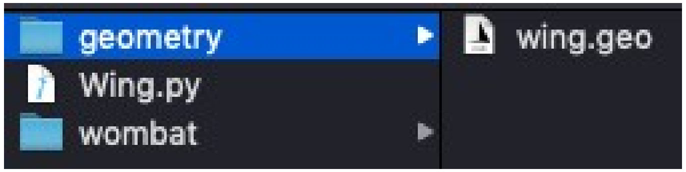
Structure of the main script

1. **Geometry description and mesh generation** (implies prior definition of .geo)

```python
# Geometry description and mesh generation
mesh, boundary = call_gmsh('geometry/wing.geo',SolidT3)
right = boundary.get_elem_from_tag("right").get_nodes()
left = boundary.get_elem_from_tag("left").get_nodes()
```

Type of element required

Located in the geometry folder



Input.py

```python
def call_gmsh(geofile,elem_type):
    """ Calls ``Gmsh`` to generate a mesh

    Parameters
    ----------
    geofile : str
        path to GEO file, must end in ".geo"
    elem_type : {:class:`SolidT3`,:class:`SolidT6`}
        type of generated elements

    Returns
    -------
    regions
        a :class:`Mesh <mesh.Mesh>` object
    """
    if issubclass(elem_type,Triangle6):
        order = 2
    else:
        order = 1
    command = 'gmsh -order '+str(order)+' '+geofile+' -2'
    os.system(command)
    print "Finished meshing!\n"
    mshfile = geofile[:-3]+'msh'
    return read_msh(mshfile,elem_type)
```

Modify path in call_gmsh method.
(here for instance for a MacOs)

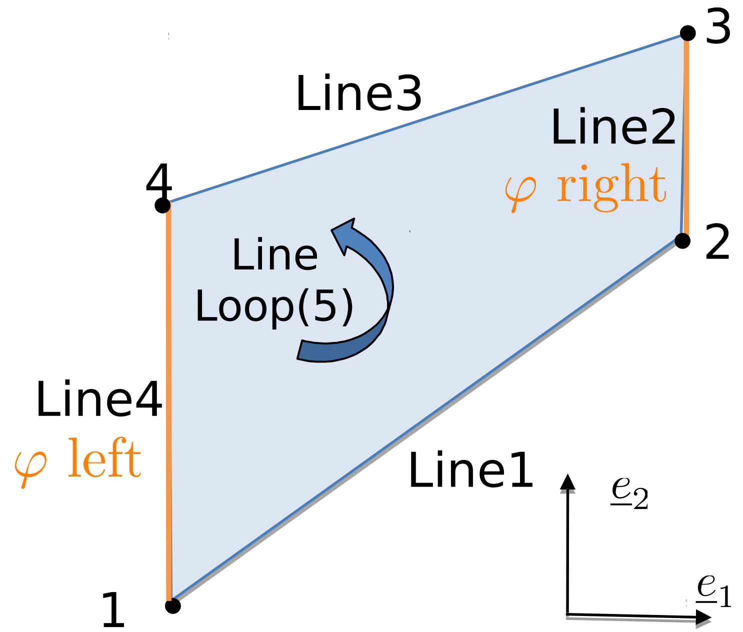
```python
    command = '/Applications/Gmsh.app/Contents/MacOS/gmsh -order '+str(order)+' '+geofile+' -2'
```

Structure of the main script

1. Geometry description and mesh generation (implies prior definition of .geo)

```
// global dimensions
L1 = 48.0;
L2 = 44.0;
L3 = 60.0;

// mesh density
lc1 = 30.;

// points coordinates
Point(1) = {0.0,0.0,0.0,lc1};
Point(2) = {L1,L2,0.0,lc1};
Point(3) = {L1,L3,0.0,lc1};
Point(4) = {0.0,L2,0.0,lc1};

// lines and line loops
Line(1) = {1,2};
Line(2) = {2,3};
Line(3) = {3,4};
Line(4) = {4,1};
Line Loop(5) = {1,2,3,4};

// surface
Plane Surface(1) = {5};

// physical entities
Physical Line("right") = {2};
Physical Line("left") = {4};
Physical Surface("wing") = {1};
```

wing.geo



Physical Line = where BC are applied

Physical Surface = where material properties are assigned

Structure of the main script

wing.msh

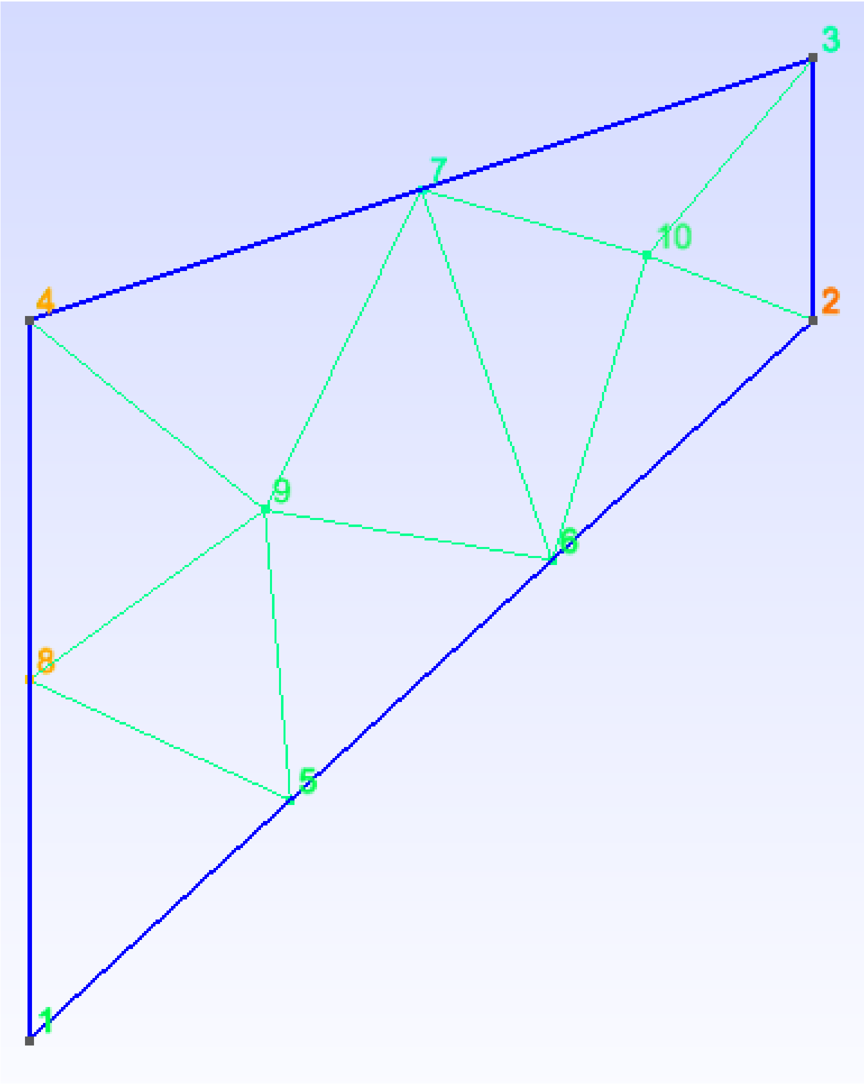
```
$MeshFormat
2.2 0 8
$EndMeshFormat
$PhysicalNames
3
1 1 "left"
1 2 "right”
2 3 "wing"
$EndPhysicalNames
$Nodes
10
1 0 0 0
2 48 44 0
3 48 60 0
4 0 44 0
5 15.99999999994201 14.66666666661351 0
6 31.99999999994015 29.33333333327847 0
7 24.00000000005996 52.00000000001999 0
8 0 22.00000000004024 0
9 14.41534391534047 32.40404040399452 0
10 37.80952380953335 47.95238095237472 0
$EndNodes
…
```

(Dim, n° Phys. Entity, name)

Total number of nodes

~ Coordinates matrix

(n° node, X, Y, Z)



Structure of the main script

wing.msh

…
$Elements
13 ← Total number of elements

1 1 2 1 2 2 3
2 1 2 2 4 4 8
3 1 2 2 4 8 1
4 2 2 3 1 1 5 8
5 2 2 3 1 2 10 6
6 2 2 3 1 4 9 7
7 2 2 3 1 6 10 7
8 2 2 3 1 4 8 9
9 2 2 3 1 6 7 9
10 2 2 3 1 5 6 9
11 2 2 3 1 5 9 8
12 2 2 3 1 2 3 10
13 2 2 3 1 3 7 10

$EndElements

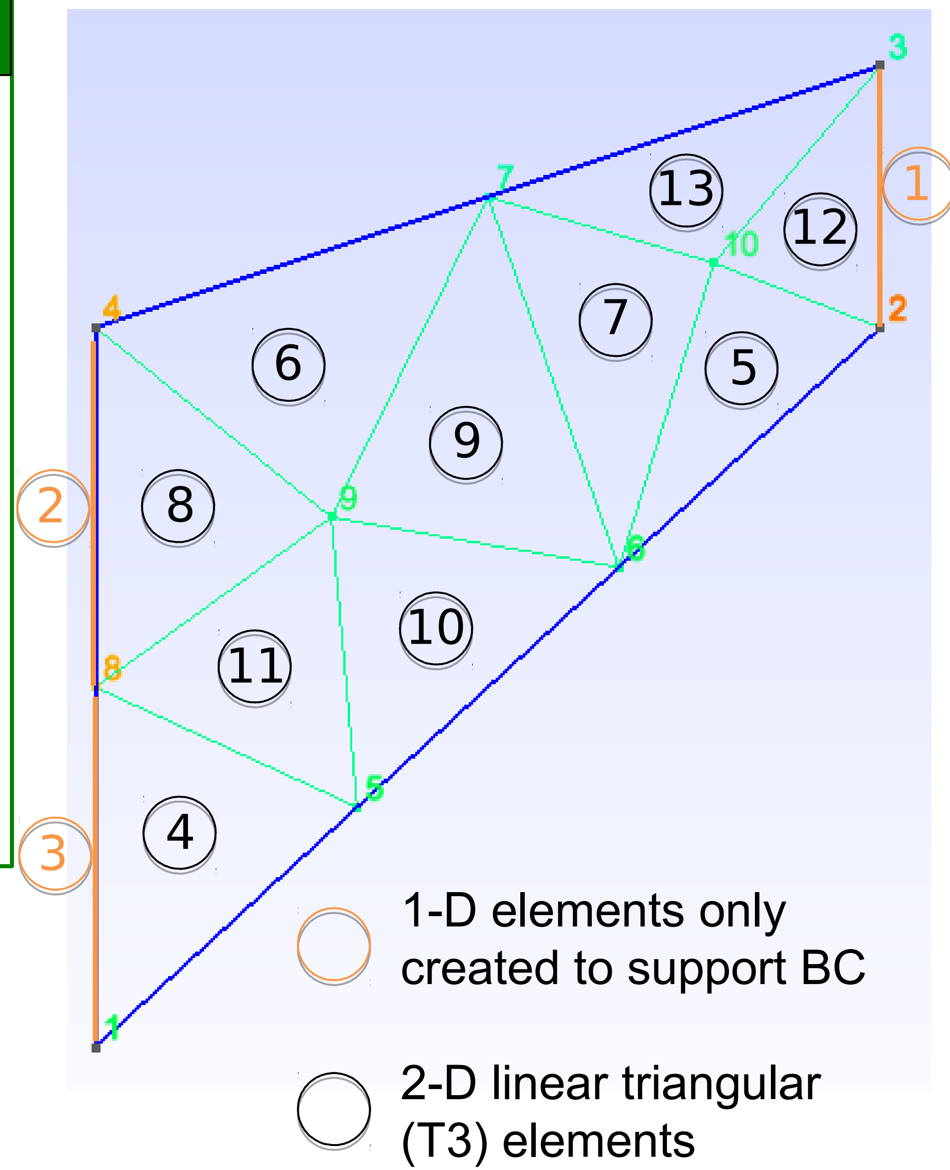
~ Connectivity matrix
(n° node,Type,…,
Phys. Set, Orig. Geom. Set,
Nodes connectivity)

Type:

- 1=2-node line
- 2=3-node triangle
- etc

Ex : 6 2 2 3 1 4 9 7

- Elt n°: 6
- Type: 2
- Dimension of analysis: 2
- Physical set: 3
- Original Geometrical set: 1
- Nodes connectivity: 4 9 7



Structure of the main script

1. **Material properties** (computation of the elasticity tensor,…)

```
# Material properties
mat_concr = LinearElastic(E=Yg_mod,nu=Nu_poi,model="plane_stress")
```

material.py

```
class LinearElastic(Material):
    """ Linear elastic material

    Attributes
    ----------
    Young_modulus : float
        material Young modulus :math:`E`
    Poisson_coeff : float
        material Poisson coefficient :math:`\\nu` (with :math:`-1<\\nu<1/2`),
        ignored for :class:`Bar2D <bar2D.Bar2D>`  and :class:`Beam2D <beam2D.Beam2D>`  elements
    rho : float
        material volumetric mass density :math:`\\rho`
    model : {'plane_strain','plane_stress'}
        type of 2D model
    C : ndarray
        elasticity matrix :math:`[C]` shape=(3,3)
    """
```

Available Plane elasticity models (no axisymmetry so far…)

Structure of the main script

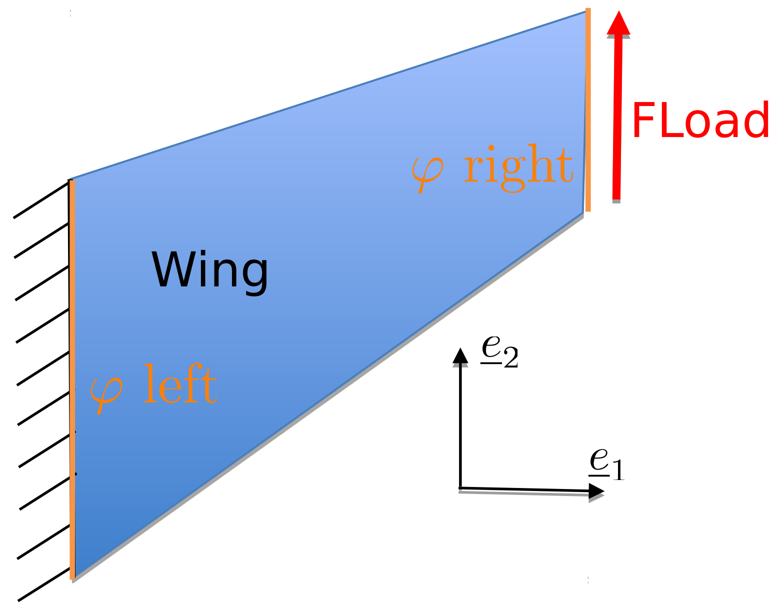
1. **Boundary conditions and load**

```
# Boundary conditions and load
appuis = Connections()
appuis.add_imposed_displ(left,ux=0,uy=0)

forces = ExtForce()
forces.add_distributed_forces(right,fy=FLoad)
```

See : **connections.py** & **forces.py**

Possibility to define :

- **Imposed displacements or imposed relations** between nodes displacements (master/slaves relationships)
- **concentrated and distributed forces** (and **couples** for beam elements only)



Structure of the main script

1. **Model construction** (assigning materials properties to the corresponding mesh regions,)

```python
# Model construction
model = Model(mesh,mat)
```

model.py

2. **Assembly phase** (construction of the stiffness matrix [K] and second member {F}, BC taken into account)

```python
# Assembly phase
K = assembl_stiffness_matrix(model)
L,Ud = assembl_connections(appuis,model)
F = assembl_external_forces(forces,model)
```

finite_elements_sparse.py

Structure of the main script

3. Solving (of [K]{U}={F})

```
# Solving
U,lamb = solve(K,F,L,Ud)
```

finite_elements_sparse.py

Augmented Lagrangian solving method
U = displacements at nodes
lamb = lambda = Lagrange multipliers = Reaction forces components

4. Post-processing

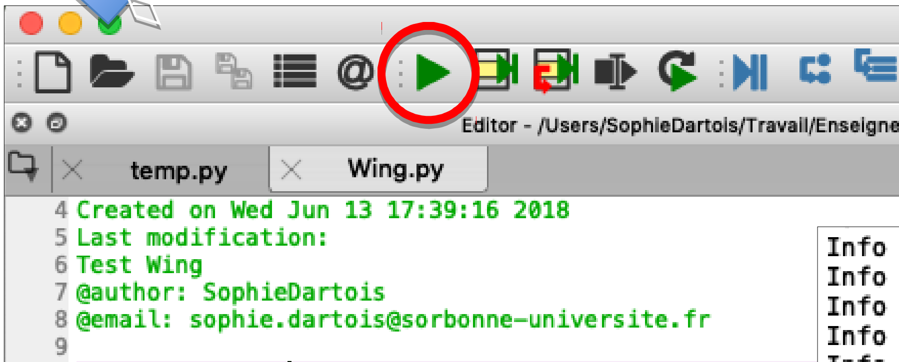
```
# Post-processing
coeff = 100
list_res = post_proc_res()
list_res.compute_res(U,model)
list_res.trace_res(mesh,coeff,U,appuis,mat.model)
```

res_treat.py

Displacement computation, stresses (compute_res),
and display as isovalues (trace_res)

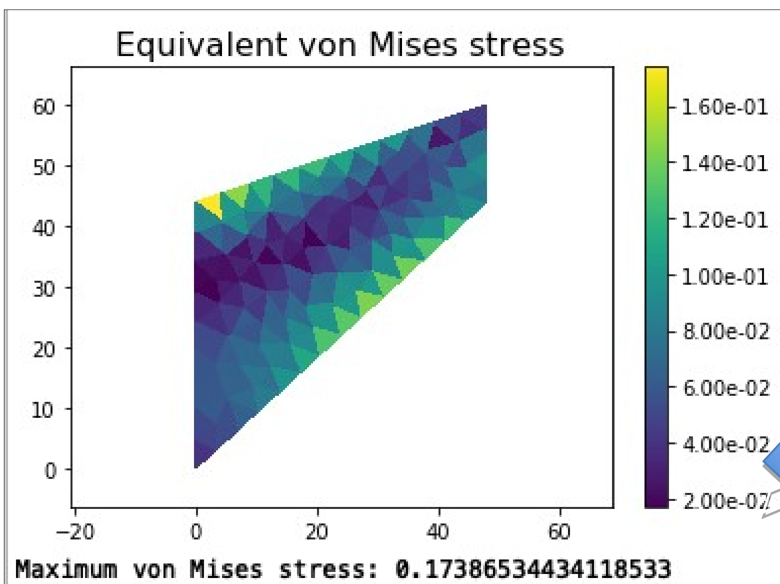
Running the code

1- In Spyder, run the script Wing.py by clicking on the « run » icon for instance :



2- You should see infos from gmsh indicating that meshing was performed successfully

```
Info    : Started on Wed Jul 24 16:30:08 2019
Info    : Reading 'geometry/wing.geo'...
Info    : Done reading 'geometry/wing.geo'
Info    : Finalized high order topology of periodic connections
Info    : Meshing 1D...
Info    : Meshing curve 1 (Line)
Info    : Meshing curve 2 (Line)
Info    : Meshing curve 3 (Line)
Info    : Meshing curve 4 (Line)
Info    : Done meshing 1D (0.000975 s)
Info    : Meshing 2D...
Info    : Meshing surface 1 (Plane, Delaunay)
Info    : Done meshing 2D (0.00553918 s)
Info    : 103 vertices 208 elements
Info    : Writing 'geometry/wing.msh'...
Info    : Done writing 'geometry/wing.msh'
Info    : Stopped on Wed Jul 24 16:30:08 2019
Finished meshing!
```



3- In the end you should see in the console isovalue graphs asked for as well as any information needing to be calculated and displayed (such as max VM stress here)