Finite element analysis (FEA) with python

1 Short introduction to the FEA code

The finite element analysis (FEA) code used is written in Python 2.7. It's a semi-interpreted programming language which means it doesn't have to be compiled but scripts can be directly executed from the console, for instance with the following command line:
>:python Name_of_my_pgrm.py

1.1 Quick overview of the language (Python 2.7)

It is to be noted that this is not a proper lesson about Python 2.7 but only a list of definitions and concepts that you will need to know and understand in order to use the code.

1.1.1 General points

It is recommended that you install Anaconda distribution : `https://www.anaconda.com/`.
It can be seen as a "box" containing several python related softwares such as Spyder (graphic environment), Jupyter,...

Every single script begins with the following type of sentence:

```
# -*- coding:  utf-8 -*-
```

It is indeed mandatory to set the coding type of the script (utf-8, iso-8859-15,...).

Python :

- is **case-sensitive**, ie. Myname ≠ MyName.
- must have a single instruction on each line (otherwise use "`\`" at the end of each unfinished line.)
- is **very sensitive to indentation**
- allows comments : `#` for comments on a single line, `"""bla bla"""`, or `'''bla bla'''` otherwise (for comments on several lines for instance).

1.1.2 Main existing types for variables

- **Numbers:** `int`, `long`, `float`, `complex` with a `j` (ex: `3+4j`)

- **Non-Editable sequences:**

- Strings (`str`), or defined like this `'text'` or `"text"`
- Tuples (between brackets) $\simeq$ non editable list such as `('a','b','c')`

- **Editable sequences:** What you would commonly refer to as an array and which is a `list` in python such as: `['a','b','c']`;

  - Access to the elements of the list is granted with the following syntax:

```
>>>Mylist = [3, 21, 5]
>>>Mylist[1]
21
>>>Mylist[0]
3
>>>Mylist[-1]
5
```

⚠ **Warning : numbering starts with zero ! ex: `Mylist[0]`**
It can be reversed (ex: index -1, -2, ...)

  - Slices can be extracted `[start:stop:step]`, step =1 if omitted.

```
>>>Mylist = [0, 1, 2, 3, 4, 5]
>>>Mylist[1:3]
[1, 2, 3]
>>>Mylist[-2:]
[4, 5]
>>>Mylist[::-1]
[5, 4, 3, 2, 1]
```

  - Modify one component or more with:

```
Mylist[i] = x
Mylist[i:j:k] = t
```

  - Delete one component or more with:

```
del Mylist[i]
Mylist[i] = []
del Mylist[i:j:k]
```

  - Add one component or more with:

```
Mylist.append(x)
Mylist.insert(i,x)
Mylist.extend(t)
```

  - Other operations : `reverse()`, `sort()`

- **Editable associations:**
  - dictionaries (`dict`) : `{'a':'val', 3:'x', 'key':124}`
  - sets (`set`) : `{'a', 3, 'key'}`

- **Other types:** booleans (`bool`), non editable associations (`frozenset`), ...

Note : the corresponding transformation functions exist, such as `str()`, `int()`, `float()`, ...

1.1.3 Definition of classes and methods

Classes in python (`class`) are objects with potentially several *attributes*. Attributes are variables that can be of different types and which can be given an initial value. They are reached thanks to the operator "." as shown below:

```
>>>class Vector:
        x = 3
        y = 4
>>> v = Vector()
>>> v.x, v.y
(3, 4)
```

An object may contain other objects. Distinct levels are reached with series of ".", example:

```
Object.subobject.component
```

Methods are special functions dedicated to a class and defined in the class. Their first parameters are always the current instance (aka `self`).

```
>>>class Vector:
        x = 3
        y = 4
        def norm(self):
            x2 = self.x**2
            y2 = self.y**2
            return (x2+y2)**(1/2)
```

Two specific methods must now be introduced: *constructors* and *destructors*.

- The constructor `__init__()` is called when a new object is created. It is a private method which is why it is surrounded by `__`:

```
>>>class Vector:
        def __init__(self):
            self.components = []
        def norm(self):
            x2 = self.components[0]**2
            y2 = self.components[1]**2
            return (x2+y2)**(1/2)
```

- The destructor `__del__()` is self explanatory...

2 A (very) short introduction to WomBat

The code is actually a python library known as WomBat, kindly provided to you and developed by Jeremy Bleyer Ã l'ENPC (jeremy.bleyer@enpc.fr).

2.1 Creation of the Master script

In order to solve a structure problem with this code you will need to write a python script which will be refered to as the *Master script*. Open Spyder and create a new script. Give it a revealing name while saving it (something like *Flexion_of_pipe.py*).

The first line of your Master script will traditionally be:

```
# -*- coding:  utf-8 -*-
```

Then you are encouraged as usual to write a few lines between """ describing the content of your Master script, including the date and the author so that future developer may reach you.

```
"""
In this script is solved a flexural problem on a pipe...bla bla
Created on June 11th 2018 @last modification:  June 13th 2018
@author:  Paul Duchemin
@email:  paul.duchemin@etu.sorbonne-universite.fr
"""
```

Next step is to import the WomBat library thanks to the following command:

```
from wombat import *
```

Your script must now contain all the common steps to perform FEA :

1. Creation of the geometry
2. Mesh generation
3. Definition of material properties
4. Loading and boundary conditions
5. Assembly phase
6. Actual solving
7. Post-processing

2.2 Geometry and mesh

Only 2D structures will be studied with this code so only a small number of geometrical will be used here.

Nodes

A single class of nodes is here defined: `class Nodes2D`.
It has one attribute which is the list of its coordinates.

```
>>> n0 = Node2D([2., 0.])
>>> n0.coor
>>> array([2., 0.])
```

Elements library

In this code each element is actually a class defined in a short python script located in the *element* directory. As a consequence the geometry (ex: 2-node segment) and kinematics (isoparametric or subparametric) of elements are defined simultaneously.
You will find below an alphabetical list of the classes of elements together with a short description.

Bar2D	a unidirectional truss element with 2 dof (u_x, u_y) in a 2D space.
Beam2D	a unidirectional beam element with Navier-Euler-Bernoulli kinematics and 3 dof (u_x, u_y, θ_z)
SolidT3	a surface triangular element, dof (u_x, u_y, θ_z)
SolidT6	a surface triangular element, dof (u_x, u_y, θ_z)

Other files contain methods (*__init__.py*, *generic_element.py* and *trace_elements.py*) for all the classes of elements.

Mesh generation

Nodes and elements can be directly defined in the Master script for simple geometries. For complex meshes a call to GMSH[1] mesher is required. The python script `input.py` contains functions allowing to import meshes from Gmsh.

2.3 Methods dictionary for all the other steps:

As it would be too long to describe each class and corresponding methods allowing to perform FEA simulations only a glossary will be provided here. In order to decipher the code it is recommended to look for the comments section at the beginning of each file or class. Note that constructors are omitted for a better legibility.

[1] http://gmsh.info/

File	Classes	Methods
connections.py	Connections	add_relation
		add_imposed_displ
	YieldLineConnections	add_imposed_displ
finite_elements_sparse.py		assembl_connections
		assembl_stiffness_matrix
		assembl_external_forces
		assembl_thermal_strains
		solve
		stresses
		assembl_initial_state
forces.py	ExtForce	add_distributed_forces
		add_concentrated_forces
		add_constant_pressure
geometric_caract.py	Section	
	BeamSection	rect
input.py		call_gmsh
		read_msh
material.py	Material	
	LinearElastic	compute_lame_coeff
		from_lame
		C_matrix
		compute_sigzz
mesh.py	Mesh	volume
		print_info
model.py	Model	affect_property
node.py	Node	get_dof
	Node2D	copy
	NodeGroup	add_node
		get_dof
		get_id
post_process.py	Figure	show
		clear
		add_subplot
		plot
		plot_bc
		plot_def
		plot_field
		plot_field_diagrams
		axes_rescale
res_treat	post_proc_res	compute_res
		trace_res
utils.py		append_component
		uniquify
		value_at_next_step