oogmsh Python package, User's Guide[*]

version 0.1.0

F. Cuvelier[†]

May 11, 2019

Abstract

The oogmsh Python package make it possible by using **gmsh** to generate mesh files (MSH format version 2.2, 4.0 or 4.1) from *.geo* file. It's also possible with the `ooGmsh2` class or with the `ooGmsh4` class to read MSH file respectively in version 2.2 and versions 4.x. This toolbox must be regarded as a very simple interface between gmsh files and Python. So you are free to create any objects you want from an `ooGmsh2` object or an `ooGmsh4` object.

0 Contents

[*] LaTeX manual, revision 0.1.0, compiled with Python 3.6.8, and toolboxes **fc-oogmsh**[0.1.0], **fc-tools**[0.0.22], **fc-meshtools**[0.1.0], **fc-matplotlib4mesh**[0.1.0], **fc-mayavi4mesh**[0.1.0],

[†] LAGA, UMR 7539, CNRS, Université Paris 13 - Sorbonne Paris Cité, Université Paris 8, 99 Avenue J-B Clément, F-93430 Villetaneuse, France, cuvelier@math.univ-paris13.fr

1 Introduction

The fc_oogmsh Python toolbox is closely related to **gmsh**, see [4] or [5], which is a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. **gmsh** can also build two-dimensional meshes and three-dimensional surface meshes. This toolbox was initialy created to make it possible from Python to rapidly

- generate mesh file from .geo file by using **gmsh**
- efficiently read this mesh file and store its contents in ooGmsh Python object easy to manipulate.

The `ooGmsh` Python object can be used to create, from a .msh file, any objects needed by your project. For example, the fc-simesh Python toolbox uses this toolbox to create the siMesh object containing all the simplices elements of the mesh.

This package is also provided with some simple graphics tools using the FC-MATPLOTLIB4MESH package [1] or FC-MAYAVI4MESH package [3].

This package was tested under

OS	Python	gmsh
CentOS 7.6	[7] 2.7.16, 3.5.7, 3.6.8, 3.7.3	4.3.0, 4.2.3, 4.1.5, 4.0.7, 3.0.6
Debian 9.9	[7] 2.7.16, 3.5.7, 3.6.8, 3.7.3	4.3.0, 4.2.3, 4.1.5, 4.0.7, 3.0.6
Ubuntu 18.04 LTS	[7] 2.7.16, 3.5.7, 3.6.8, 3.7.3	4.3.0, 4.2.3, 4.1.5, 4.0.7, 3.0.6
OpenSuse 15.0	[7] 3.5.7, 3.6.8, 3.7.3	4.3.0, 4.2.3, 4.1.5, 4.0.7, 3.0.6
Fedora 29	[7] 2.7.16, 3.5.7, 3.6.8, 3.7.3	4.3.0, 4.2.3, 4.1.5, 4.0.7, 3.0.6
MacOS Mojave 10.14.4	[7]: 3.5.4, 3.6.8, 3.7.2	4.3.0, 4.2.3, 4.1.5, 4.0.7, 3.0.6
Windows 10 (1809)	[7]: 3.6.8, 3.7.3	4.3.0, 4.2.3, 4.1.5, 4.0.7, 3.0.6

In a first step we quickly present the installation and the configuration of the fc_oogmsh toolbox for using the **gmsh** application. Thereafter, we describe the fc_oogmsh's functions which use **gmsh** to create mesh files.

2 Installation and configuration

2.1 Installation

- For an installation without graphic tools which isolated to the current user, one can do:

```
$ pip install --user fc_oogmsh
```

- For an installation without graphic tools for all users, one can do:

```
$ sudo pip install fc_oogmsh
```

For Python 3 installation, `pip3` must be used instead of `pip`.

To enable the graphic tools one has to install at least one of these packages. For example, to use the FC-MATPLOTLIB4MESH package:

```
$ pip install --user fc_oogmsh[matplotlib4mesh]
```

One can also install the two packages FC-MATPLOTLIB4MESH and FC-MAYAVI4MESH :

```
$ pip install --user fc_oogmsh[matplotlib4mesh,mayavi4mesh]
```

2.2 Configuration

One have to configure the package for using with **gmsh**. For the default configuration we run under Python:

```
>>> import fc_oogmsh
>>> fc_oogmsh.configure()
```

By default, the gmsh binary is supposed to be located in

- `<USERDIR>/bin/gmsh` under linux,
- `<USERDIR>/GMSH/Gmsh.app/Contents/MacOS/gmsh` under Mac OS X,
- `<USERDIR>/Softwares/GMSH/gmsh.exe` under Windows

The `fc_oogmsh.configure()` try to guess where is the **gmsh** binary. If this command failed or if we want to specify the **gmsh** binary location, one can use the **gmsh** option to specify the **gmsh** binary file with full path.

- For example, under Linux:

  ```
  >>> fc_oogmsh.configure(gmsh='/usr/local/GMSH/gmsh-3.0.4-Linux/bin/gmsh')
  ```

- For example, under Windows:

  ```
  >>> fc_oogmsh.configure(gmsh=r'C:\Users\toto\GMSH\gmsh-3.0.4-Windows\gmsh.exe')
  ```

  Note that the **gmsh** string is given as *raw litteral string* (`r'...'`) to avoid using of escape sequences as `'\n'`, `'\t'`, ...

- For example, under MacOs:

  ```
  >>> fc_oogmsh.configure(gmsh='/Users/toto/GMSH/3.0.4/Gmsh.app/Contents/MacOS/gmsh')
  ```

Now, it's possible to run one of the demo functions

Python code with output

```
fc_oogmsh.demo02();
```

```
**********************
Running demo02 function
**********************
*** Build mesh file
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/condenser11.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
*** Read mesh file
*** Print oGh ->
ooGmsh4 object
    dim : 2
      d : 2
  types : [1 2]
     nq : 3483
      q : ndarray object[float64], size (2, 3483)
toGlobal: ndarray object[int32], size (3483,)
Entities:<class 'fc_oogmsh.msh.Entities'>
Nodes   :<class 'fc_oogmsh.msh.Nodes'>
Elements:<class 'fc_oogmsh.msh.Elements'>
```

3 gmsh interface functions

All the functions provided in this section use **gmsh** to create a mesh file from a **gmsh** geometry script file (extension *.geo*).

3.1 function `fc_oogmsh.buildmesh2d`

This function uses **gmsh** and a *.geo* file (describing a 2D-geometry) to generate a 2D-mesh.

Syntaxe

```
meshfile=fc_oogmsh.buildmesh2d(geofile,N)

meshfile=fc_oogmsh.buildmesh2d(geofile,N,Key=Value)
```

Description

`meshfile=fc_oogmsh.buildmesh2d(geofile,N)` create a 2D-mesh using **gmsh** and the *geo* file `geofile`. The integer `N` has two functions : numbering the name of the generated mesh as <geofile without extension and path> + <-N.msh> and passing this number to **gmsh** via the option "-setnumber N <N>". Usually we used this parameter in **gmsh** to set the prescribed mesh element size at the points. (see given *geo* files)
As output return a file name (with full path) corresponding to the mesh generated by **gmsh**.

`meshfile=fc_oogmsh.buildmesh2d(geofile,N,Key=Value, ...)` specifies function options using one or more `Key`,`Value` pair arguments. The `Key` options can be

- `meshdir` : to specify the directory where the mesh file will be written, (default: output of the function `fc_oogmsh.Sys.getDefaultMeshDir()`)
- `meshfile` : to specify the name of the mesh file. Without directory path, the mesh file is saved in directory given by `meshdir` option.
(default: `<meshdir>/<meshfile>-<N>.msh`)
- `force` : to force meshing even if the mesh file already exists if `Value` is `true` (default : `false`)
- `verbose` : to specify the degree of verbosity (0, silence; 1, default; 2, command and **gmsh** output)
- `options` : string which contains command-line options used with **gmsh** (default empty). For example, one can use `options='-string␣"Mesh.Algorithm=1;"␣-string␣"Mesh.ScalingFactor=2;"'` (see **gmsh** documentation)
- `MshFileVersion` : to specify the MSH file format version. `Value` could be
  - `'2.2'` if gmsh version ⩾ 3.0.0,
  - `'4.0'` if gmsh version ⩾ 4.0.0,
  - `'4.1'` if gmsh version ⩾ 4.2.0.

Usage of the `N` parameter To illustrate the usage of the `N` parameter, the `square.geo` file, used to mesh the square $[0, L] \times [0, L]$, is given in Listing 1. In this file, the `N` parameter is used to set the **gmsh** parameter `Mesh.CharacteristicLengthFactor`. As we can see in the output of the Python command, the number of vertices in the mesh increase with `N` value.

```
DefineConstant[
  N = {10, Name "Input/1Refine param."},
  L = {1., Name "Input/2Side length"}
];
Mesh.CharacteristicLengthFactor=5*L/N;
Point (1) = {0, 0, 0};
Point (2) = {L, 0, 0};
Point (3) = {L, L, 0};
Point (4) = {0, L, 0};
Line (3) = {1, 2};
Line (2) = {2, 3};
Line (4) = {3, 4};
Line (1) = {4, 1};
Line Loop (100) = { 1, 2,  3,  4};
Plane Surface (1) = {100};
Physical Line(1) = {1};
Physical Line(2) = {2};
Physical Line(3) = {3};
Physical Line(4) = {4};
Physical Surface(1) = {1};
Physical Point(101) = {1};
Physical Point(102) = {2};
Physical Point(103) = {3};
Physical Point(104) = {4};
```

Listing 1: square.geo file

Python code with output

```python
meshfile=fc_oogmsh.buildmesh2d('geofiles/square.geo',20,force=True)
oGh1=fc_oogmsh.ooGmsh4(meshfile)
print('Generated␣mesh␣file:\n␣␣->␣%s'%meshfile)
print('␣␣␣␣->␣number␣of␣vertices:␣%d'%oGh1.nq)
meshfile=fc_oogmsh.buildmesh2d('geofiles/square.geo',200,
    force=True,verbose=2)
oGh2=fc_oogmsh.ooGmsh4(meshfile)
print('Generated␣mesh␣file:\n␣␣->␣%s'%meshfile)
print('␣␣␣␣->␣number␣of␣vertices:␣%d'%oGh2.nq)
```

```
[fc_oogmsh] Using input file: geofiles/square.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/square-20.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
Generated mesh file:
  -> /home/cuvelier/.local/share/fc_oogmsh/meshes/square-20.msh
    -> number of vertices: 1156
[fc_oogmsh] Using input file: geofiles/square.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/square-200.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
[fc_oogmsh] Command line:
  /home/cuvelier/bin/gmsh -2 -setnumber N 200 -string " Mesh.MshFileVersion=4.1;" "geofiles/square.geo" -o
         "/home/cuvelier/.local/share/fc_oogmsh/meshes/square-200.msh"
[fc_oogmsh] Running command. Be patient...
Generated mesh file:
  -> /home/cuvelier/.local/share/fc_oogmsh/meshes/square-200.msh
    -> number of vertices: 106563
```

An another way is given in Listing 2.

```
DefineConstant[
  N = {10, Name "Input/1Refine param."},
  L = {1., Name "Input/2Side length"}
];
h=L/N;
Point (1) = {0, 0, 0, h};
Point (2) = {L, 0, 0, h};
Point (3) = {L, L, 0, h};
Point (4) = {0, L, 0, h};
Line (3) = {1, 2};
Line (2) = {2, 3};
Line (4) = {3, 4};
Line (1) = {4, 1};
Line Loop (100) = { 1, 2,  3,  4};
Plane Surface (1) = {100};
Physical Line(1) = {1};
Physical Line(2) = {2};
Physical Line(3) = {3};
Physical Line(4) = {4};
Physical Surface(1) = {1};
Physical Point(101) = {1};
Physical Point(102) = {2};
Physical Point(103) = {3};
Physical Point(104) = {4};
```

Listing 2: square2.geo file

Python code with output

```
meshfile=fc_oogmsh.buildmesh2d('geofiles/square2.geo',20, force=True)
oGh1=fc_oogmsh.ooGmsh4(meshfile)
print('Generated␣mesh␣file:\n␣␣->␣%s'%meshfile)
print('␣␣␣␣->␣number␣of␣vertices:␣%d'%oGh1.nq)
meshfile=fc_oogmsh.buildmesh2d('geofiles/square2.geo',200,
    force=True,verbose=2)
oGh2=fc_oogmsh.ooGmsh4(meshfile)
print('Generated␣mesh␣file:\n␣␣->␣%s'%meshfile)
print('␣␣␣␣->␣number␣of␣vertices:␣%d'%oGh2.nq)
```

```
[fc_oogmsh] Using input file: geofiles/square2.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/square2-20.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
Generated mesh file:
  -> /home/cuvelier/.local/share/fc_oogmsh/meshes/square2-20.msh
    -> number of vertices: 568
[fc_oogmsh] Using input file: geofiles/square2.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/square2-200.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
[fc_oogmsh] Command line:
  /home/cuvelier/bin/gmsh -2 -setnumber N 200 -string " Mesh.MshFileVersion=4.1;" "geofiles/square2.geo" -o
         "/home/cuvelier/.local/share/fc_oogmsh/meshes/square2-200.msh"
[fc_oogmsh] Running command. Be patient...
Generated mesh file:
  -> /home/cuvelier/.local/share/fc_oogmsh/meshes/square2-200.msh
    -> number of vertices: 53302
```

Examples All the following examples use the *.geo* file `condenser11.geo` which is in the directory `geodir/2d` of the toolbox.

Python code with output

```
print('***␣fc_oogmsh.buildmesh2d␣:␣1st␣call')
meshfile=fc_oogmsh.buildmesh2d('condenser11',25,force=True)
print('***␣fc_oogmsh.buildmesh2d␣:␣2nd␣call')
meshfile=fc_oogmsh.buildmesh2d('condenser11',25)
```

```
*** fc_oogmsh.buildmesh2d : 1st call
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/condenser11.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
*** fc_oogmsh.buildmesh2d : 2nd call
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/condenser11.geo
[fc_oogmsh] Mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh already exist.
  -> Use "force" flag to rebuild if needed.
```

Python code with output

```
meshfile=fc_oogmsh.buildmesh2d('condenser11',25,force=True, verbose=2,
    options='-string␣"Mesh.Algorithm=1;"␣-string␣"Mesh.ScalingFactor=2;"')
```

```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/condenser11.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
[fc_oogmsh] Command line:
  /home/cuvelier/bin/gmsh -2 -setnumber N 25 -string "Mesh.Algorithm=1;" -string "Mesh.ScalingFactor=2;" -string " Mesh.MshFileVersion=4.1;"
         "/fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/condenser11.geo" -o
         "/home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh"
[fc_oogmsh] Running command. Be patient...
```

3.2 function `fc_oogmsh.buildmesh3d`

This function uses **gmsh** and a *.geo* file (describing a 3D-geometry) to generate a 3D-mesh. See function `fc_oogmsh.buildmesh2d` for usage and options.

3.3 function `fc_oogmsh.buildmesh3ds`

This function uses **gmsh** and a *.geo* file (describing a 3D surface geometry or a 3D-geometry) to generate a 3D surface mesh. See function `fc_oogmsh.buildmesh2d` for usage and options.

3.4 function `fc_oogmsh.buildpartmesh2d`

This function uses **gmsh** and a *.msh* file (containing of a 2D-mesh) to generate and save a 2D partioned mesh. Returns the filename of the partitioned mesh.

Syntaxe

```
pmfile=fc_oogmsh.buildpartmesh2d(meshfile,np)

pmfile=fc_oogmsh.buildpartmesh2d(meshfile,np,Key=Value)
```

Description

`pmfile=fc_oogmsh.buildpartmesh2d(meshfile,np)` create a 2D partitioned mesh using `gmsh` and the *msh* file `meshfile` (with path). The integer `np` is the number of partitions.
As output return a file name (with full path) corresponding to the partitioned mesh generated by `gmsh`. The output file name is construct as following : <meshfile without extension>-part<np>.msh

`pmfile=fc_oogmsh.buildpartmesh2d(meshfile,np,Key=Value, ...)` specifies function options using one or more `Key`,`Value` pair arguments. The `Name` options can be

- `force` : to force meshing even if the mesh file already exists if `Value` is `True` (default : `false`)
- `verbose` : to specify the degree of verbosity (0, silence; 2, default; ...)
- `options` : string which contains command-line options used with gmsh (see `gmsh` documentation)

Examples All the following examples use the `meshfile` as output of the command :
`meshfile=fc_oogmsh.buildmesh2d('condenser11',25);`

Python code with output

```
print('***␣Building␣the␣mesh')
meshfile=fc_oogmsh.buildmesh2d('condenser11',25)
print('***␣Partitioning␣the␣mesh')
pmfile=fc_oogmsh.buildpartmesh(meshfile,5,force=True)
```

```
*** Building the mesh
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/condenser11.geo
[fc_oogmsh] Mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh already exist.
  -> Use "force" flag to rebuild if needed.
*** Partitioning the mesh
[fc_oogmsh] Using input file: /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25-part5.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
```

Python code with output

```python
print('***␣Building␣the␣mesh')
meshfile=fc_oogmsh.buildmesh2d('condenser11',25, verbose=1)
print('***␣Partitioning␣the␣mesh')
pmfile=fc_oogmsh.buildpartmesh(meshfile,5,force=True, verbose=3,
    options='-string␣"Mesh.MetisAlgorithm=3;"')
```

```
*** Building the mesh
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/condenser11.geo
[fc_oogmsh] Mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh already exist.
  -> Use "force" flag to rebuild if needed.
*** Partitioning the mesh
[fc_oogmsh] Using input file: /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25-part5.msh
[fc_oogmsh] Command line:
  /home/cuvelier/bin/gmsh -2 -saveall -part 5 -string "Mesh.MetisAlgorithm=3;" -string " Mesh.MshFileVersion=4.1;"
        "/home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh" -o "/home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25-part5.msh"
[fc_oogmsh] Running command. Be patient...
[fc_oogmsh] Printing command output:
Info    : Running '/home/cuvelier/bin/gmsh -2 -saveall -part 5 -string Mesh.MetisAlgorithm=3; -string Mesh.MshFileVersion=4.1;
        /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh -o /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25-part5.msh' [Gmsh 4.2.2, 1
        node, max. 1 thread]
Info    : Started on Sat May 11 08:40:47 2019
Info    : Reading '/home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh'...
Info    : 3080 nodes
Info    : 6262 elements
Info    : Done reading '/home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25.msh'
Info    : Meshing 1D...
Info    : Done meshing 1D (1.2e-05 s)
Info    : Meshing 2D...
Info    : Done meshing 2D (1.3e-05 s)
Info    : 3089 vertices 6311 elements
Info    : Partitioning mesh...
Info    : Running METIS graph partitioner
Info    : 5 partitions, 165 total edge-cuts
Info    : Done partitioning mesh (0.019025 s)
Info    :  - Repartition of 49 point(s): 6(min) 19(max) 9.8(avg)
Info    :  - Repartition of 360 line(s): 68(min) 76(max) 72(avg)
Info    :  - Repartition of 5902 triangle(s): 1180(min) 1181(max) 1180.4(avg)
Info    : Creating partition topology...
Info    :  - Creating partition edges
Info    :  - Creating partition vertices
Info    : Done creating partition topology (0.001871 s)
Info    : Writing '/home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25-part5.msh'...
Info    : Done writing '/home/cuvelier/.local/share/fc_oogmsh/meshes/condenser11-25-part5.msh'
Info    : Stopped on Sat May 11 08:40:47 2019
```

3.5 function `fc_oogmsh.buildpartmesh3d`

This function uses **gmsh** and a *.msh* file (containing of a 3D-mesh) to generate a 3D partioned mesh.

3.6 function `fc_oogmsh.buildpartmesh3ds`

This function uses **gmsh** and a *.msh* file (containing of a 3D surface mesh) to generate a 3D partioned surface mesh.

3.7 function `fc_oogmsh.buildPartRectangle`

This function uses **gmsh** and the *geodir/rectanglepart.geo* file to generate a 2D regular partioned mesh of the rectangle $[0, Lx] \times [0, Ly]$ with $Nx \times Ny$ partitions.

Syntaxe

```python
meshfile=fc_oogmsh.buildpartrectangle(Lx,Ly,Nx,Ny,N)

meshfile=fc_oogmsh.buildpartrectangle(Lx,Ly,Nx,Ny,N, Key=Value)
```

Description

`meshfile=fc_oogmsh.buildpartrectangle(Lx,Ly,Nx,Ny,N)` create a 2D regular partitioned mesh using **gmsh** of the rectangle $[0, Lx] \times [0, Ly]$ with $Nx \times Ny$ partitions. The `N` parameter is passed to **gmsh** to set the prescribed mesh element size at the points
As output return a file name (with full path) corresponding to the partitioned mesh generated by **gmsh**. The default output file name is construct as following : rectanglepart-Lx%.3f-Ly%.3f-Nx%d-Ny%d-N%d.msh

`meshfile=fc_oogmsh.buildpartrectangle(Lx,Ly,Nx,Ny,N,Key=Value, ...)` specifies function options using one or more `Key`,`Value` pair arguments. The `Key` options can be

- force : to force meshing even if the mesh file already exists if Value is True (default : False)
- verbose : to specify the degree of verbosity (0, silence; 2, default; ...)
- options : string which contains command-line options used with gmsh (see gmsh documentation)

Examples All the following examples ...

Python code with output

```
pmfile=fc_oogmsh.buildpartrectangle(1,1,3,2,100,force=True, verbose=True)
```

```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/rectanglepart.geo
[fc_oogmsh] Use option verbose=3 to see gmsh output
```

Python code with output

```
pmfile=fc_oogmsh.buildpartrectangle(1,1,3,2,100,verbose=True,
    force=True,meshfile='./toto.msh')
```

```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/rectanglepart.geo
[fc_oogmsh] Use option verbose=3 to see gmsh output
```

4 fc_oogmsh.ooGmsh4 class

The ooGmsh4 class can be used to read gmsh mesh files with the MSH ASCII file format version 4.1 since gmsh 4.1.0 ([6], section 9.1) or version 4.0 since gmsh 4.0.0.

The gmsh's native "MSH" file format (version 4.x) is used to store meshes and associated post-processing datasets either save as an ASCII file or a binary file with extension .msh. The focus of the ooGmsh4 class is to read only meshes contained in an ASCII file. Currently, it is not planned to read post-processing datasets.

As described in [6], section 9.1: *the MSH file format version 4 (current revision: version 4.1) contains one mandatory section giving information about the file (*$MeshFormat*), followed by several optional sections defining the physical group names (*$PhysicalName*), the elementary geometrical entities (*$Entities*), the partitioned entities (*$PartitionedEntities*), the nodes *$Nodes*), the elements (*$Elements*), the periodicity relations (*$Periodic*), the ghost* elements ($GhostElements) and the post-processing datasets ($NodeData, $ElementData, $ElementNodeData).

For each section, the ooGmsh4 class has a property with corresponding name. The properties of this class are:

Properties of ooGmsh4 class

dim	:	space dimension (2 or 3)
nq	:	number of nodes/vertices.
q	:	nodes/vertices array with dimension dim-by-nq.
toGlobal	:	...
MeshFormat	:	dictionary
PhysicalNames	:	(optional), array of class fc_oogmsh.msh.PhysicalName
Entities	:	class fc_oogmsh.msh.Entities
PartitionedEntities	:	None or class fc_oogmsh.msh.PartitionedEntities
Nodes	:	class fc_oogmsh.msh.Nodes
Elements	:	class fc_oogmsh.msh.Elements

The dictionary MeshFormat, all the classes are described in section 4.2. In the following subsections, Gh is an ooGmsh4 object.

4.1 Methods

4.1.1 ooGms4 constructor

The ooGmsh4 class have only one constructor :

```
Gh=fc_oogmsh.ooGmsh4(meshfile)
Gh=fc_oogmsh.ooGmsh4(meshfile, verbose=False)
```

where `meshfile` is the name of ... a mesh file. The `verbose` Key/Value option can be used to print some informations, when reading the file `meshfile`, if `Value` is `True`. Default is `False`

Python code with output

```
print('1) Building the mesh')
meshfile=fc_oogmsh.gmsh.buildmesh2d('condenser',10, Verbose=0,force=True)
print('2) Reading the mesh')
Gh = fc_oogmsh.ooGmsh4(meshfile,verbose=True)
print('-> Gh is an ooGmsh4 object containing a MSH file version '+Gh.MeshFormat['version'])
print('3) Displaying Gh')
print(Gh)
```

```
1) Building the mesh
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/condenser.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser-10.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
2) Reading the mesh
-> Gh is an ooGmsh4 object containing a MSH file version 4.1
3) Displaying Gh
ooGmsh4 object
    dim : 2
      d : 2
  types : [ 1 2 15]
     nq : 9116
      q : ndarray object[float64], size (2, 9116)
toGlobal: ndarray object[int32], size (9116,)
Entities:<class 'fc_oogmsh.msh.Entities'>
Nodes   :<class 'fc_oogmsh.msh.Nodes'>
Elements:<class 'fc_oogmsh.msh.Elements'>
```

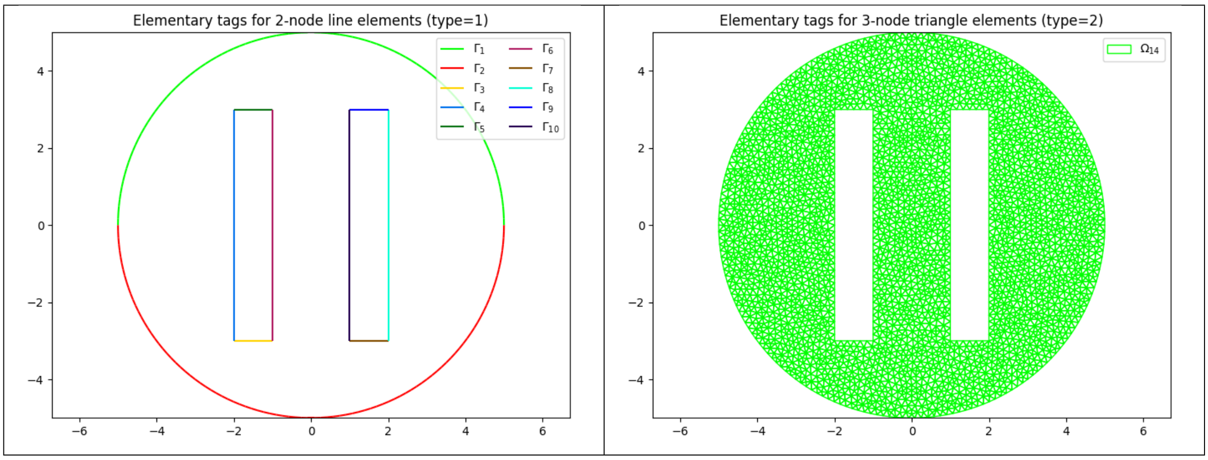


Figure 1: *Elementary Tag* elements of the *geofile* `condenser.geo`

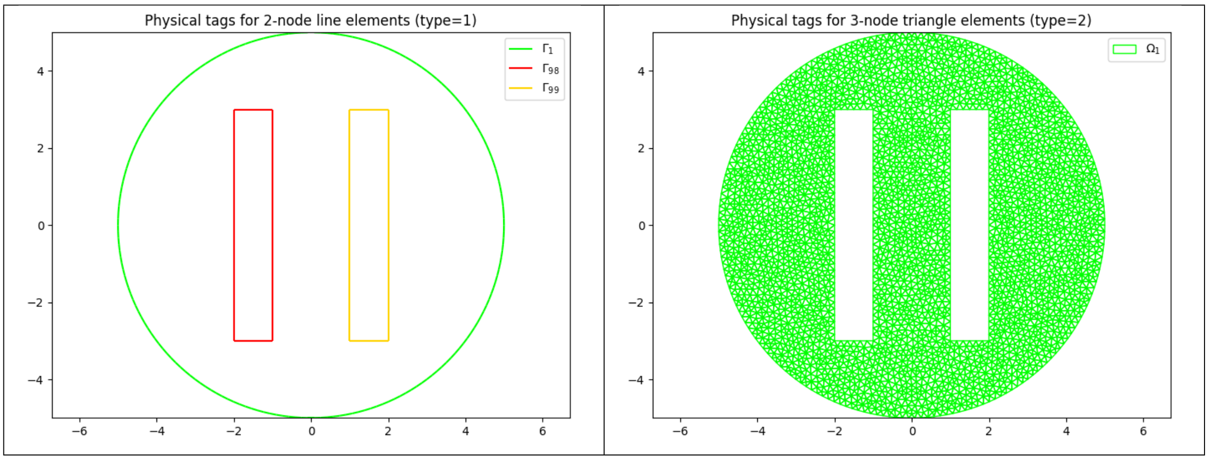


Figure 2: *Physical Tag* elements of the *geofile* `condenser.geo`

In the *geofile* `condenser.geo` the *Physical Tags* are created from the *Elementary Tags* as follow

```
...
Physical Line(1) = {1, 2};
Physical Line(98) = {5, 6, 3, 4};
Physical Line(99) = {9, 8, 7, 10};
Physical Surface(1) = {14};
```

4.1.2 `get_ElementaryTags` method

```
eltags=Gh.get_ElementaryTags(EltType)
```

Description

`eltags=Gh.get_ElementaryTags(EltType)`
returns all the elementary tags associated with elements of type `EltType` as an array with unique elements. `EltType` is described in Appendix A. For example, `EltType` is 1 for `2-nodes line` (i.e 1-simplex of order 1), `EltType` is 2 for `3-nodes triangle` (i.e 2-simplex of order 1) and `EltType` is 4 for `4-nodes tetrahedron` (i.e 3-simplex of order 1).

Python code with output

```
eltags1=Gh.get_ElementaryTags(1)
print('eltags1='+str(eltags1))
eltags2=Gh.get_ElementaryTags(2)
print('eltags2='+str(eltags2))
```

```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/condenser.geo
[fc_oogmsh] Mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser-6.msh already exist.
  -> Use "force" flag to rebuild if needed.
eltags1=[ 1 2 3  4  5  6  7  8  9 10]
eltags2=[14]
```

4.1.3 `get_PhysicalTags` method

```
phtags=Gh.get_PhysicalTags(EltType)
```

Description

`phtags=Gh.get_PhysicalTags(EltType)`
returns all the elementary tags associated with elements of type `EltType` as an array with unique elements.

Python code with output

```
phtags1=Gh.get_PhysicalTags(1)
print('phtags1='+str(phtags1))
phtags2=Gh.get_PhysicalTags(2)
print('phtags2='+str(phtags2))
```

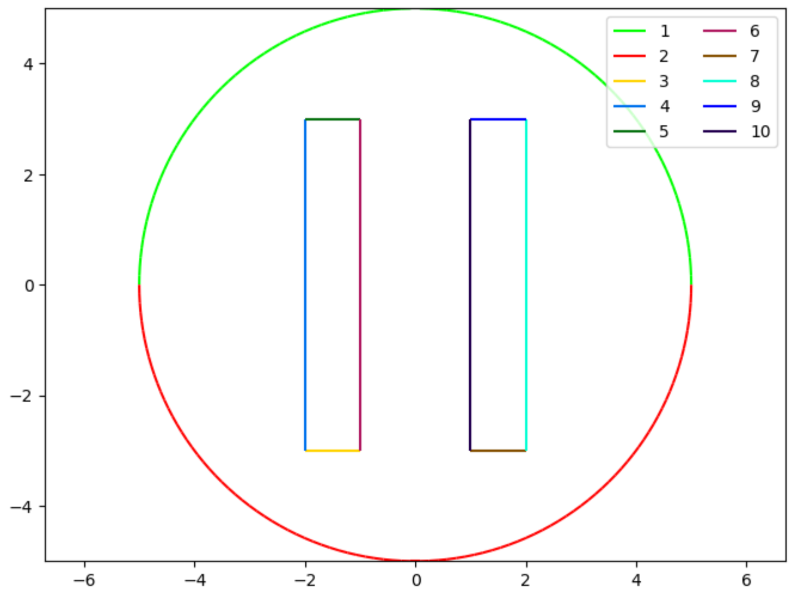
```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/condenser.geo
[fc_oogmsh] Mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser-6.msh already exist.
  -> Use "force" flag to rebuild if needed.
phtags1=[ 1 98 99]
phtags2=[1]
```

4.1.4 `get_me_ElementaryTag` method

```
me=Gh.get_me_ElementaryTag(EltType,EltTag)
```

Description

`me=Gh.get_me_ElementaryTag(EltType,EltTag)`
returns `me` the connectivity array of mesh elements of type and *elementary tag* given respectively by `EltType` and `EltTag`. This array is associated with the `Gh.q` nodes/vertices array.




```
meshfile=fc_oogmsh.gmsh.buildmesh2d('condenser',6)
Gh = fc_oogmsh.ooGmsh4(meshfile)
import matplotlib.pyplot as plt
from fc_tools.Matplotlib import set_axes_equal,set_axes
import fc_matplotlib4mesh.simplicial as plt4sim
from fc_tools.colors import selectColors
eltags1=Gh.get_ElementaryTags(1)
n1=len(eltags1)
colors = selectColors(n1)
plt.figure(1)
Legend=[];Labels=[]
for i in range(n1):
  me=Gh.get_me_ElementaryTag(1,eltags1[i])
  pm=plt4sim.plotmesh(Gh.q,me,color=colors[i])
  Legend.append(pm)
  Labels.append(str(eltags1[i]))
set_axes_equal()
plt.legend(Legend,Labels,loc='best', ncol=int(len(Legend)/10)+1)
set_axes(plt.gca(),Gh.bbox)
```

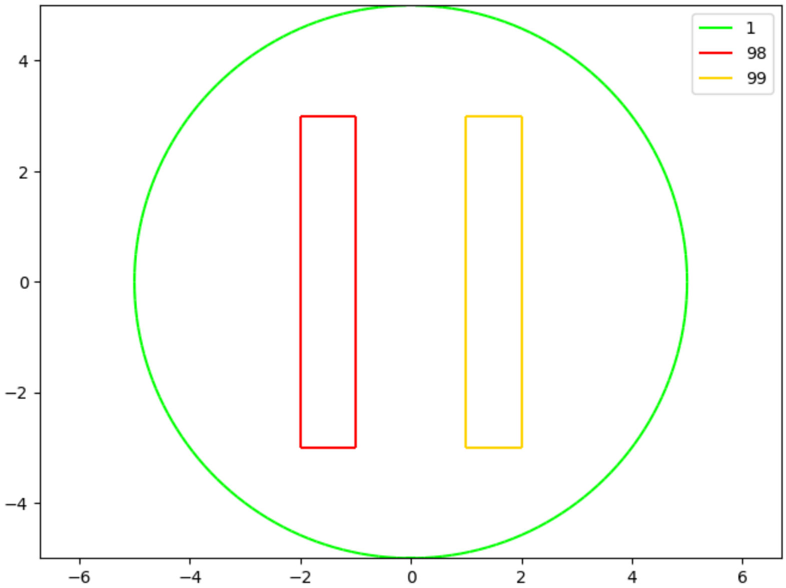
Listing 3: Plot curves mesh elements by using **get_me_ElementaryTag** function

4.1.5 `get_me_PhysicalTag` method

```
me=Gh.get_me_PhysicalTag(EltType,PhyTag)
```

Description

`me=Gh.get_me_PhysicalTag(EltType,PhyTag)`
returns `me` the connectivity array of mesh elements of type and *physical tag* given respectively by `EltType` and `PhyTag`. This array is associated with the `Gh.q` nodes/vertices array.



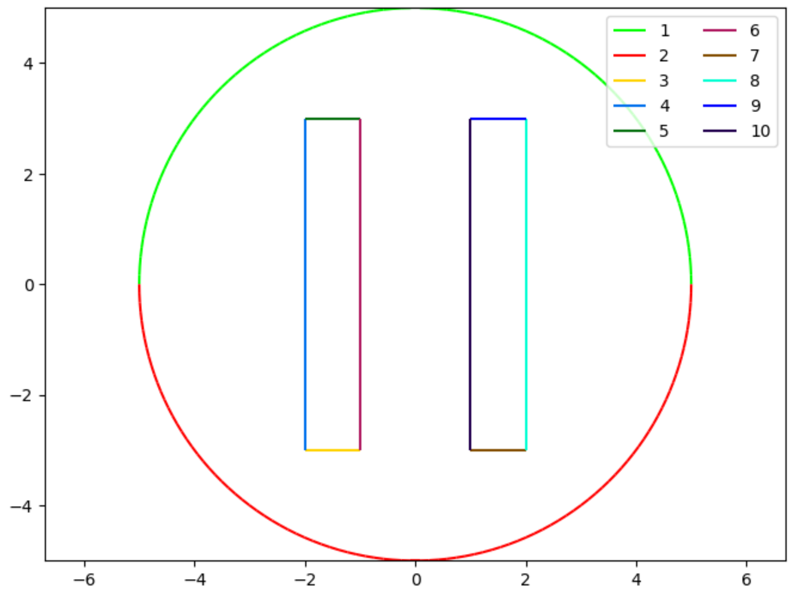

```
meshfile=fc_oogmsh.gmsh.buildmesh2d('condenser',6)
Gh = fc_oogmsh.ooGmsh4(meshfile)
import matplotlib.pyplot as plt
from fc_tools.Matplotlib import set_axes_equal,set_axes
import fc_matplotlib4mesh.simplicial as plt4sim
from fc_tools.colors import selectColors
phtags1=Gh.get_PhysicalTags(1)
n1=len(phtags1)
colors = selectColors(n1)
plt.figure(1)
Legend=[];Labels=[]
for i in range(n1):
  me=Gh.get_me_PhysicalTag(1,phtags1[i])
  pm=plt4sim.plotmesh(Gh.q,me,color=colors[i])
  Legend.append(pm)
  Labels.append(str(phtags1[i]))
set_axes_equal()
plt.legend(Legend,Labels,loc='best', ncol=int(len(Legend)/10)+1)
set_axes(plt.gca(),Gh.bbox)
```

Listing 4: Plot curves mesh elements by using **get_me_PhysicalTag** function

4.1.6 `get_localmesh_ElementaryTag` method

```
q,me,toGlobal=Gh.get_localmesh_ElementaryTag(EltType,EltTag)
```

`q,me,toGlobal=Gh.get_localmesh_ElementaryTag(EltType,EltTag)`
returns the *local* nodes/vertices array `q` and the *local* connectivity array `me` of the element of type `EltType` and with *elementary tag* given by `EltTag`. The *global* tags array `toGlobal` is such that `Gh.q(:,toGlobal)` is equal to `q`.



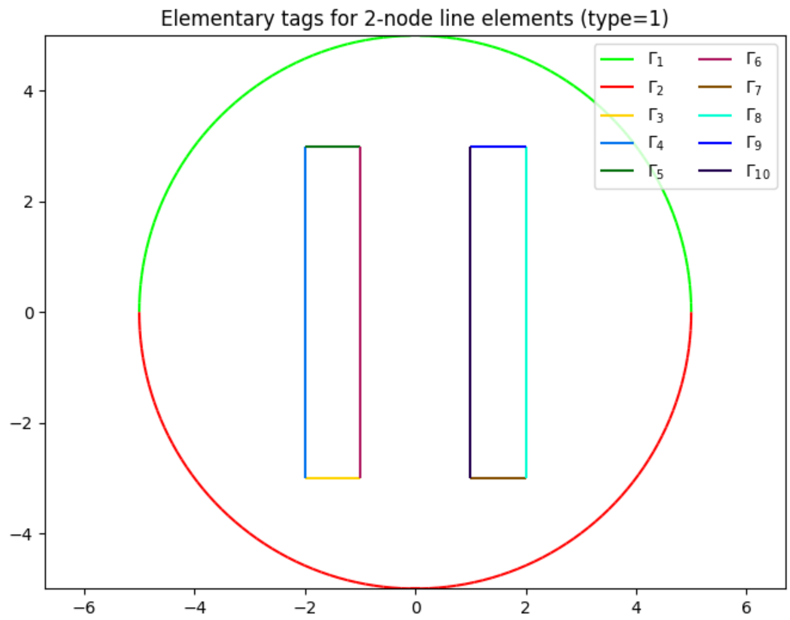

```
meshfile=fc_oogmsh.gmsh.buildmesh2d('condenser',6)
Gh = fc_oogmsh.ooGmsh4(meshfile)
import matplotlib.pyplot as plt
from fc_tools.Matplotlib import set_axes_equal,set_axes
import fc_matplotlib4mesh.simplicial as plt4sim
from fc_tools.colors import selectColors
eltags1=Gh.get_ElementaryTags(1)
n1=len(eltags1)
colors = selectColors(n1)
plt.figure(1)
Legend=[];Labels=[]
for i in range(n1):
  q,me=Gh.get_localmesh_ElementaryTag(1,eltags1[i])[:2]
  pm=plt4sim.plotmesh(q,me,color=colors[i])
  Legend.append(pm)
  Labels.append(str(eltags1[i]))
set_axes_equal()
plt.legend(Legend,Labels,loc='best', ncol=int(len(Legend)/10)+1)
set_axes(plt.gca(),Gh.bbox)
```

Listing 5: Plot *2-nodes line* mesh elements by using **get_localmesh_ElementaryTag** function

4.1.7 `get_localmesh_PhysicalTag` method

```
q,me,toGlobal=Gh.get_localmesh_PhysicalTag(EltType,PhysicalTag)
```

`q,me,toGlobal=Gh.get_localmesh_PhysicalTag(EltType,PhyTag)`
returns the *local* nodes/vertices array `q` and the *local* connectivity array `me` of the elements of type `EltType` and with *PhyTag* given by `PhysicalTag`. The *global* tags array `toGlobal` is such that `Gh.q(:,toGlobal)` is equal to `q`.




```
meshfile=fc_oogmsh.gmsh.buildmesh2d('condenser',6)
Gh = fc_oogmsh.ooGmsh4(meshfile)
import matplotlib.pyplot as plt
from fc_tools.Matplotlib import set_axes_equal,set_axes
import fc_matplotlib4mesh.simplicial as plt4sim
from fc_tools.colors import selectColors
phtags1=Gh.get_PhysicalTags(1)
n1=len(phtags1)
colors = selectColors(n1)
plt.figure(1)
Legend=[];Labels=[]
for i in range(n1):
  q,me=Gh.get_localmesh_PhysicalTag(1,phtags1[i])[:2]
  pm=plt4sim.plotmesh(q,me,color=colors[i])
  Legend.append(pm)
  Labels.append(str(phtags1[i]))
set_axes_equal()
plt.legend(Legend,Labels,loc='best', ncol=int(len(Legend)/10)+1)
set_axes(plt.gca(),Gh.bbox)
```

Listing 6: Plot *2-nodes line* mesh elements by using **get_localmesh_PhysicalTag** function

4.2 Description of properties

Keys of `MeshFormat` dictionary

`version`	:	string, version of the mesh file format.
`file_type`	:	integer, 0 for ASCII mode, 1 for binary mode.
`data_size`	:	integer, `sizeof(size_t)`

Properties of (optional) `fc_oogmsh.msh.PhysicalName` class

`dimension`	:	integer.
`physicalTag`	:	integer.
`name`	:	string

Properties of `fc_oogmsh.msh.Entities` class

`numPoints`	:	integer.
`Points`	:	array of `Point` class.
`numCurves`	:	integer.
`Curves`	:	array of `Curve` class.
`numSurfaces`	:	integer.
`Surfaces`	:	array of `Surface` class.
`numVolumes`	:	integer.
`Volumes`	:	array of `Volume` class.

Properties of (optional) `fc_oogmsh.msh.PartitionedEntities` class

`numPartitions`	:	integer.
`numGhostEntities`	:	integer.
`GhostEntities`	:	array of class.
`numPoints`	:	integer
`Points`	:	array of class.
`numCurves`	:	integer
`Curves`	:	array of class.
`numSurfaces`	:	integer
`Surfaces`	:	array of class.
`numVolumes`	:	integer.
`Volumes`	:	array of class.

Properties of `fc_oogmsh.msh.Nodes` class

`numEntityBlocks`	:	integer.
`numNodes`	:	integer.
`minNodeTag`	:	integer.
`maxNodeTag`	:	integer
`EntityBlocks`	:	array of `fc_oogmsh.msh.NodesEntityBlock` class.

Properties of `fc_oogmsh.msh.NodesEntityBlock` class

`entityTag`	:	integer.
`entityDim`	:	integer.
`parametric`	:	integer.
`numNodes`	:	integer.
`nodeTags`	:	1-by-`numNodes` array of integer.
`Nodes`	:	3-by-`numNodes` array of double.

Properties of the `fc_oogmsh.msh.Elements` class

`numEntityBlocks`	:	integer.
`numElements`	:	integer.
`minElementTag`	:	integer.
`maxElementTag`	:	integer
`EntityBlocks`	:	array of `fc_oogmsh.msh.ElementsEntityBlock` class.
`ElementTypes`	:	array of .

Properties of the `fc_oogmsh.msh4_1.ElementsEntityBlock` class

`entityDim`	:	integer.
`entityTag`	:	integer.
`elementType`	:	integer.
`elementDesc`	:	structure returned by the function `get_elm_desc_by_type(elementType)` of the `fc_oogmsh.gmsh` module.
`numElementsBlock`	:	integer.
`nodeTags`	:	n-by-`numElementsBlock` array. n depends of `elementType`: n = `elementDesc['nb_nodes']`
`elementTags`	:	1-by-`numElementsBlock` array

5 `fc_oogmsh.ooGmsh2` class

The `fc_oogmsh.ooGmsh2` class can be used to read **gmsh** mesh files with the MSH ASCII file format (version 2.2) described for example in [5], section 9.1. A MSH file can contain various mesh elements which are identified by an *elm-type* integer given in Appendix A. One can also refer to the `fc_oogmsh.gmsh.get_elm_desc_by_type` function, described in Appendix B.2, to obtain information on a given *elm-type*.

When reading a .msh file generated by **gmsh**, we split the mesh elements by *elm-type* and generate an array of Elmt object. The dimension of this array is the number of distinct *elm-type*s founds on the .msh file. The properties of the `Elmt` object are:

Properties of `Elmt` object

type	:	integer refers to the type of the element : 1 for 2-node line, 2 for 3-node triangle, ... See the *elm-type* description of [5], section 9.1.
geo	:	string contains the kind of geometry: 'line', 'triangle', 'tetrahedron', ...
d	:	integer space dimension or d-simplex.
order	:	integer order of the element
$\mathrm{n_{me}}$	:	integer number of mesh elements
me	:	array of $d+1$-by-$\mathrm{n_{me}}$ integers connectivity array
phys_lab	:	array of $\mathrm{n_{me}}$-by-... integers physical labels of the elements
geo_lab	:	array of $\mathrm{n_{me}}$-by-... integers geometrical labels of the elements
nb_parts	:	array of $\mathrm{n_{me}}$-by-1 integers number of mesh partitions to which the element belongs
part_lab	:	array of $\mathrm{n_{me}}$-by-max(nb_parts) integers part_lab(i, 1 : nb_parts(i)) contains all the partitions index to which the i-th element belongs.

The `fc_oogmsh.ooGmsh2` class was created to store a maximum of(all the) information(s) contained in the .msh file. The properties of this class are:

Properties of `fc_oogmsh.ooGmsh2` class

dim	:	integer space dimension
n_q	:	integer number of vertices/nodes
q	:	dim-by-n_q array of reals array of vertex coordinates
types	:	array of integers List of the element types found in the mesh file.
orders	:	array of integers List of the orders of the element types found in the mesh file.
sElts	:	array of `Elmt` objets One `Elmt` objet by element type, such that sElts(i) contains all the elements of type types(i) and order orders(i).

5.1 Methods

5.1.1 `ooGmsh2` constructor

The `fc_oogmsh.ooGmsh2` class have only one constructor :

```
Gh=fc_oogmsh.ooGmsh2(meshfile)
```

where meshfile is the name of ... a mesh file

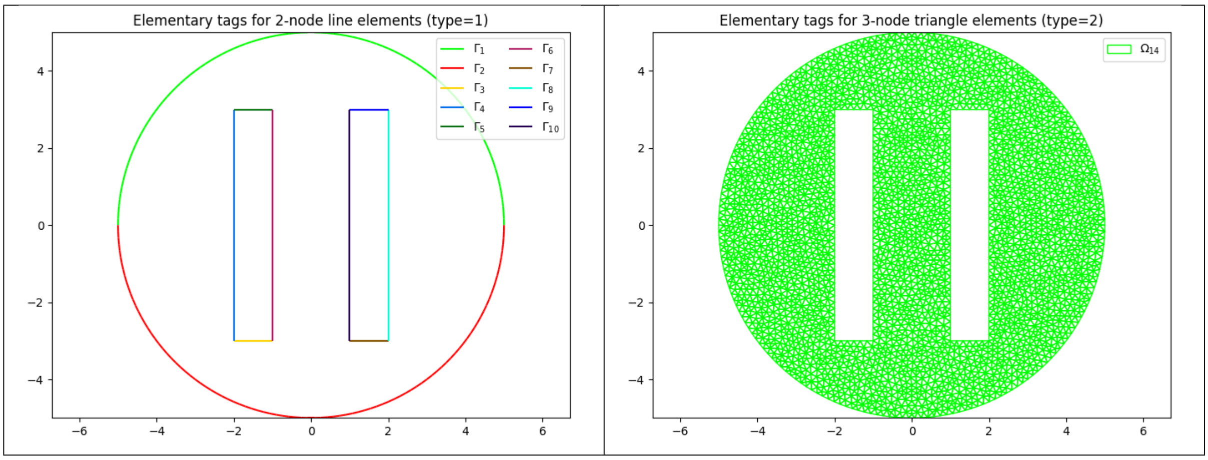


Figure 3: *Elementary Tag* elements of the *geofile* `condenser.geo`

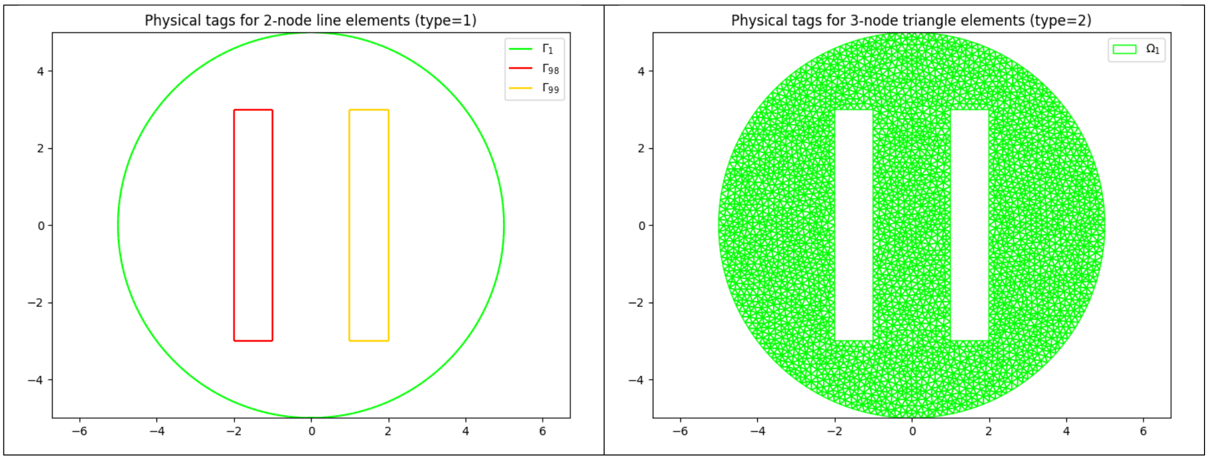


Figure 4: *Physical Tag* elements of the *geofile* `condenser.geo`

In the *geofile* `condenser.geo` the *Physical Tags* are created from the *Elementary Tags* as follow

```
...
Physical Line(1) = {1, 2};
Physical Line(98) = {5, 6, 3, 4};
Physical Line(99) = {9, 8, 7, 10};
Physical Surface(1) = {14};
```

5.1.2 `get_ElementaryTags` method

```
eltags=Gh.get_ElementaryTags(EltType)
```

Description

`eltags=Gh.get_ElementaryTags(EltType)`
returns all the elementary tags associated with elements of type `EltType` as an array with unique elements. `EltType` is described in Appendix A. For example, `EltType` is 1 for `2-nodes line` (i.e 1-simplex of order 1), `EltType` is 2 for `3-nodes triangle` (i.e 2-simplex of order 1) and `EltType` is 4 for `4-nodes tetrahedron` (i.e 3-simplex of order 1).

Python code with output

```
eltags1=Gh.get_ElementaryTags(1)
print('eltags1='+str(eltags1))
eltags2=Gh.get_ElementaryTags(2)
print('eltags2='+str(eltags2))
```

```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/condenser.geo
[fc_oogmsh] Mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser-6.msh already exist.
  -> Use "force" flag to rebuild if needed.
eltags1=[ 1 2 3  4  5  6  7  8  9 10]
eltags2=[14]
```

5.1.3 `get_PhysicalTags` method

```
phtags=Gh.get_PhysicalTags(EltType)
```

Description

`phtags=Gh.get_PhysicalTags(EltType)`
returns all the elementary tags associated with elements of type `EltType` as an array with unique elements.

Python code with output

```
phtags1=Gh.get_PhysicalTags(1)
print('phtags1='+str(phtags1))
phtags2=Gh.get_PhysicalTags(2)
print('phtags2='+str(phtags2))
```

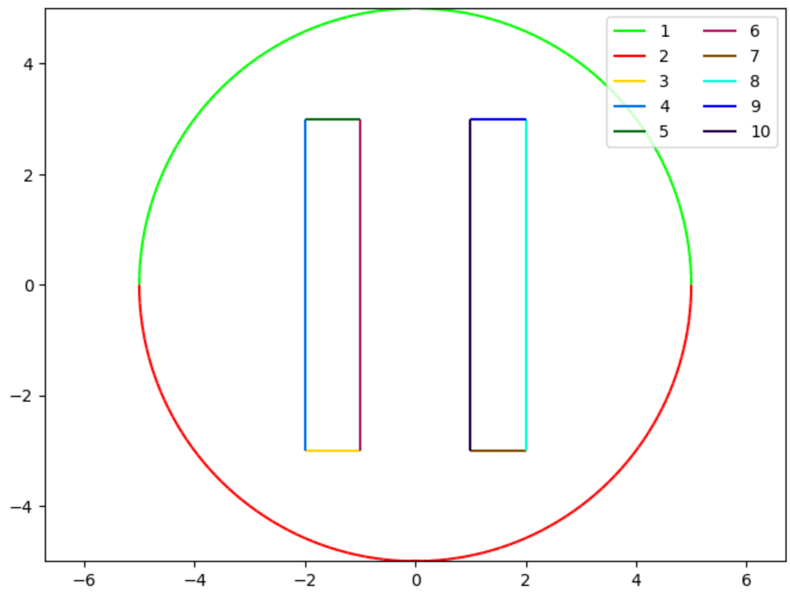
```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/condenser.geo
[fc_oogmsh] Mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser-6.msh already exist.
  -> Use "force" flag to rebuild if needed.
phtags1=[ 1 98 99]
phtags2=[1]
```

5.1.4 `get_me_ElementaryTag` method

```
me=Gh.get_me_ElementaryTag(EltType,EltTag)
```

Description

`me=Gh.get_me_ElementaryTag(EltType,EltTag)`
returns `me` the connectivity array of mesh elements of type and *elementary tag* given respectively by `EltType` and `EltTag`. This array is associated with the `Gh.q` nodes/vertices array.




```
meshfile=fc_oogmsh.gmsh.buildmesh2d('condenser',6, Force=True,MshFileVersion='2.2')
Gh = fc_oogmsh.ooGmsh2(meshfile)
import matplotlib.pyplot as plt
from fc_tools.Matplotlib import set_axes_equal,set_axes
import fc_matplotlib4mesh.simplicial as plt4sim
from fc_tools.colors import selectColors
eltags1=Gh.get_ElementaryTags(1)
n1=len(eltags1)
colors = selectColors(n1)
plt.figure(1)
Legend=[];Labels=[]
for i in range(n1):
  me=Gh.get_me_ElementaryTag(1,eltags1[i])
  pm=plt4sim.plotmesh(Gh.q,me,color=colors[i])
  Legend.append(pm)
  Labels.append(str(eltags1[i]))
set_axes_equal()
plt.legend(Legend,Labels,loc='best', ncol=int(len(Legend)/10)+1)
set_axes(plt.gca(),Gh.bbox)
```

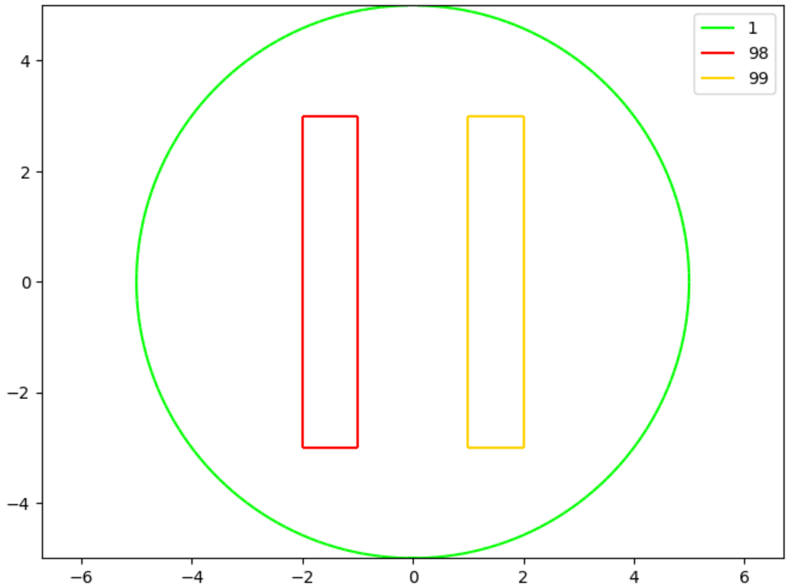
Listing 7: Plot curves mesh elements by using **get_me_ElementaryTag** function

5.1.5 `get_me_PhysicalTag` method

```
me=Gh.get_me_PhysicalTag(EltType,PhyTag)
```

Description

`me=Gh.get_me_PhysicalTag(EltType,PhyTag)`
returns `me` the connectivity array of mesh elements of type and *physical tag* given respectively by `EltType` and `PhyTag`. This array is associated with the `Gh.q` nodes/vertices array.



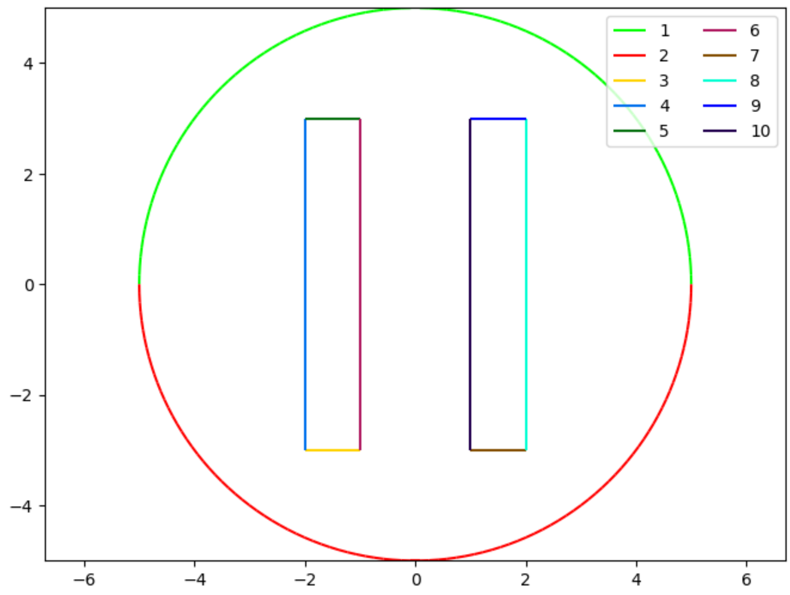

```
meshfile=fc_oogmsh.gmsh.buildmesh2d('condenser',6, Force=True,MshFileVersion='2.2')
Gh = fc_oogmsh.ooGmsh2(meshfile)
import matplotlib.pyplot as plt
from fc_tools.Matplotlib import set_axes_equal,set_axes
import fc_matplotlib4mesh.simplicial as plt4sim
from fc_tools.colors import selectColors
phtags1=Gh.get_PhysicalTags(1)
n1=len(phtags1)
colors = selectColors(n1)
plt.figure(1)
Legend=[];Labels=[]
for i in range(n1):
  me=Gh.get_me_PhysicalTag(1,phtags1[i])
  pm=plt4sim.plotmesh(Gh.q,me,color=colors[i])
  Legend.append(pm)
  Labels.append(str(phtags1[i]))
set_axes_equal()
plt.legend(Legend,Labels,loc='best', ncol=int(len(Legend)/10)+1)
set_axes(plt.gca(),Gh.bbox)
```

Listing 8: Plot curves mesh elements by using **get_me_PhysicalTag** function

5.1.6 `get_localmesh_ElementaryTag` method

```
q,me,toGlobal=Gh.get_localmesh_ElementaryTag(EltType,EltTag)
```

`q,me,toGlobal=Gh.get_localmesh_ElementaryTag(EltType,EltTag)`
returns the *local* nodes/vertices array `q` and the *local* connectivity array `me` of the element of type `EltType` and with *elementary tag* given by `EltTag`. The *global* tags array `toGlobal` is such that `Gh.q(:,toGlobal)` is equal to `q`.



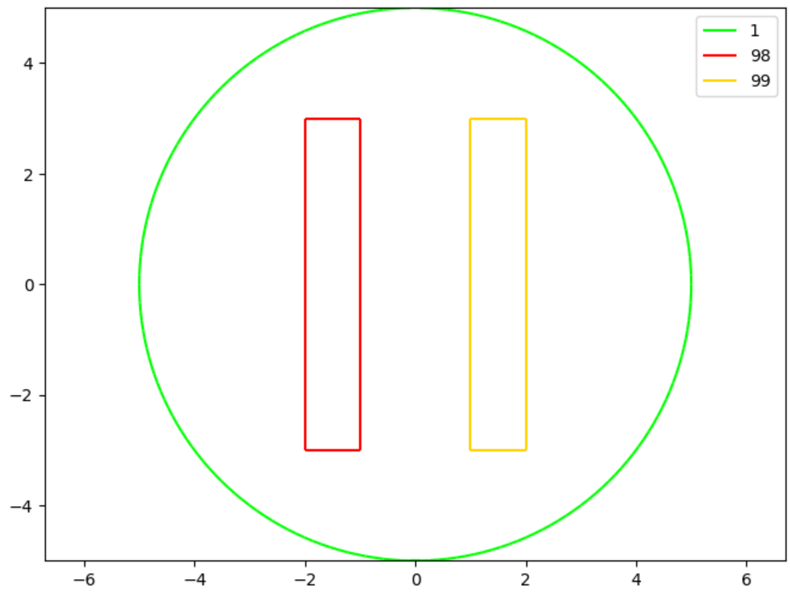

```
meshfile=fc_oogmsh.gmsh.buildmesh2d('condenser',6, Force=True,MshFileVersion='2.2')
Gh = fc_oogmsh.ooGmsh2(meshfile)
import matplotlib.pyplot as plt
from fc_tools.Matplotlib import set_axes_equal,set_axes
import fc_matplotlib4mesh.simplicial as plt4sim
from fc_tools.colors import selectColors
eltags1=Gh.get_ElementaryTags(1)
n1=len(eltags1)
colors = selectColors(n1)
plt.figure(1)
Legend=[];Labels=[]
for i in range(n1):
  q,me=Gh.get_localmesh_ElementaryTag(1,eltags1[i])[:2]
  pm=plt4sim.plotmesh(q,me,color=colors[i])
  Legend.append(pm)
  Labels.append(str(eltags1[i]))
set_axes_equal()
plt.legend(Legend,Labels,loc='best', ncol=int(len(Legend)/10)+1)
set_axes(plt.gca(),Gh.bbox)
```

Listing 9: Plot *2-nodes line* mesh elements by using **get_localmesh_ElementaryTag** function

5.1.7 `get_localmesh_PhysicalTag` method

```
q,me,toGlobal=Gh.get_localmesh_PhysicalTag(EltType,PhysicalTag)
```

`q,me,toGlobal=Gh.get_localmesh_PhysicalTag(EltType,PhyTag)`
returns the *local* nodes/vertices array `q` and the *local* connectivity array `me` of the elements of type `EltType` and with *PhyTag* given by `PhysicalTag`. The *global* tags array `toGlobal` is such that `Gh.q(:,toGlobal)` is equal to `q`.




```
meshfile=fc_oogmsh.gmsh.buildmesh2d('condenser',6, Force=True,MshFileVersion='2.2')
Gh = fc_oogmsh.ooGmsh2(meshfile)
import matplotlib.pyplot as plt
from fc_tools.Matplotlib import set_axes_equal,set_axes
import fc_matplotlib4mesh.simplicial as plt4sim
from fc_tools.colors import selectColors
phtags1=Gh.get_PhysicalTags(1)
n1=len(phtags1)
colors = selectColors(n1)
plt.figure(1)
Legend=[];Labels=[]
for i in range(n1):
  q,me=Gh.get_localmesh_PhysicalTag(1,phtags1[i])[:2]
  pm=plt4sim.plotmesh(q,me,color=colors[i])
  Legend.append(pm)
  Labels.append(str(phtags1[i]))
set_axes_equal()
plt.legend(Legend,Labels,loc='best', ncol=int(len(Legend)/10)+1)
set_axes(plt.gca(),Gh.bbox)
```

Listing 10: Plot *2-nodes line* mesh elements by using **get_localmesh_PhysicalTag** function

5.2 Samples

5.2.1 Sample 2

The 2d .geo file *condenser.geo* is used to create a .msh file : `condenser-25.msh`. This `.msh` file contains only 1 (2-node line) and 2 (3-node triangle) *elm-type*.

Python code with output

```
meshfile=fc_oogmsh.buildmesh2d('condenser',25,MshFileVersion='2.2',
    force=True)
Gh = fc_oogmsh.ooGmsh2(meshfile)
print('***␣Gh:')
print(Gh)
print('***␣Gh.sElts[0]:')
print(Gh.sElts[0])
```

```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/2d/condenser.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/condenser-25.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
*** Gh:
ooGmsh2 object
    dim : 2
      d : 2
  types : [ 1 2 15]
 orders : [1]
     nq : 55688
      q : ndarray object[float64], size (2, 55688)
toGlobal: ndarray object[int32], size (55688,)
  sElts : list of 3 elements
*** Gh.sElts[0]:
Elt object
      d : 1, type : 1, order : 1
    geo : line
    nme : 1486
     me : ndarray object[int64], size (2, 1486)
phys_lab: ndarray object[int64], size (1486,)
geo_lab : ndarray object[int64], size (1486,)
part_lab: list of 1486 elements
nb_parts: ndarray object[int64], size (1486,)
  nTags : list of 0 elements
```

5.2.2 Sample 2

The 3d .geo file *cylinderkey.geo* is used to create a .msh file : `cylinderkey-10.msh`. This `.msh` file contains 1 (2-node line), 2 (3-node triangle) and 4 (4-node tetrahedron) *elm-type*.

Python code with output

```
meshfile=fc_oogmsh.buildmesh3d('cylinderkey',10,MshFileVersion='2.2',
    force=True)
Gh = fc_oogmsh.ooGmsh2(meshfile)
print('***␣Gh:')
print(Gh)
print('***␣Gh.sElts[1]:')
print(Gh.sElts[1])
```

```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/3d/cylinderkey.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/cylinderkey-10.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
*** Gh:
ooGmsh2 object
    dim : 3
      d : 3
  types : [1 2 4]
 orders : [1]
     nq : 5152
      q : ndarray object[float64], size (3, 5152)
toGlobal: ndarray object[int32], size (5152,)
  sElts : list of 3 elements
*** Gh.sElts[1]:
Elt object
      d : 2, type : 2, order : 1
    geo : triangle
    nme : 6624
     me : ndarray object[int64], size (3, 6624)
phys_lab: ndarray object[int64], size (6624,)
geo_lab : ndarray object[int64], size (6624,)
part_lab: list of 6624 elements
nb_parts: ndarray object[int64], size (6624,)
  nTags : list of 0 elements
```

5.2.3 Sample 3

The 3d .geo file *sphere8surf.geo* is used to create a 3d surface .msh file : `sphere8surf-40.msh`. This `.msh` file contains 1 (2-node line), 2 (3-node triangle) and 15 (1-node point) *elm-type*.

Python code with output

```python
meshfile=fc_oogmsh.buildmesh3ds('sphere8surf',40,MshFileVersion='2.2',
    force=True)
Gh = fc_oogmsh.ooGmsh2(meshfile)
print('***␣Gh:')
print(Gh)
print('***␣Gh.sElts[0]:')
print(Gh.sElts[0])
```

```
[fc_oogmsh] Using input file: /fcopt/PYTHON/3.6.8/lib/python3.6/site-packages/fc_oogmsh/geodir/3ds/sphere8surf.geo
[fc_oogmsh] Overwritting mesh file /home/cuvelier/.local/share/fc_oogmsh/meshes/sphere8surf-40.msh
[fc_oogmsh] Use option verbose=3 to see gmsh output
*** Gh:
ooGmsh2 object
    dim : 3
      d : 2
  types : [ 1 2 15]
 orders : [1]
     nq : 23379
      q : ndarray object[float64], size (3, 23379)
toGlobal: ndarray object[int32], size (23379,)
  sElts : list of 3 elements
*** Gh.sElts[0]:
Elt object
      d : 1, type : 1, order : 1
    geo : line
    nme : 756
     me : ndarray object[int64], size (2, 756)
phys_lab: ndarray object[int64], size (756,)
geo_lab : ndarray object[int64], size (756,)
part_lab: list of 756 elements
nb_parts: ndarray object[int64], size (756,)
  nTags : list of 0 elements
```

A Element type

In a .msh file the kind of mesh elements are identified by their *elm-type* integer values :

elm-type	**description**
1	2-node line
2	3-node triangle
3	4-node quadrangle
4	4-node tetrahedron
5	8-node hexahedron
6	6-node prism
7	5-node pyramid
8	3-node second order line (2 nodes associated with the vertices and 1 with the edge)
9	6-node second order triangle (3 nodes associated with the vertices and 3 with the edges)
10	9-node second order quadrangle (4 nodes associated with the vertices, 4 with the edges and 1 with the face)
11	10-node second order tetrahedron (4 nodes associated with the vertices and 6 with the edges)
12	27-node second order hexahedron (8 nodes associated with the vertices, 12 with the edges, 6 with the faces and 1 with the volume)
13	18-node second order prism (6 nodes associated with the vertices, 9 with the edges and 3 with the quadrangular faces)
14	14-node second order pyramid (5 nodes associated with the vertices, 8 with the edges and 1 with the quadrangular face)
15	1-node point
16	8-node second order quadrangle (4 nodes associated with the vertices and 4 with the edges)
17	20-node second order hexahedron (8 nodes associated with the vertices and 12 with the edges)
18	15-node second order prism (6 nodes associated with the vertices and 9 with the edges)
19	13-node second order pyramid (5 nodes associated with the vertices and 8 with the edges)

20	9-node third order incomplete triangle (3 nodes associated with the vertices, 6 with the edges)
21	10-node third order triangle (3 nodes associated with the vertices, 6 with the edges, 1 with the face)
22	12-node fourth order incomplete triangle (3 nodes associated with the vertices, 9 with the edges)
23	15-node fourth order triangle (3 nodes associated with the vertices, 9 with the edges, 3 with the face)
24	15-node fifth order incomplete triangle (3 nodes associated with the vertices, 12 with the edges)
25	21-node fifth order complete triangle (3 nodes associated with the vertices, 12 with the edges, 6 with the face)
26	4-node third order edge (2 nodes associated with the vertices, 2 internal to the edge)
27	5-node fourth order edge (2 nodes associated with the vertices, 3 internal to the edge)
28	6-node fifth order edge (2 nodes associated with the vertices, 4 internal to the edge)
29	20-node third order tetrahedron (4 nodes associated with the vertices, 12 with the edges, 4 with the faces)
30	35-node fourth order tetrahedron (4 nodes associated with the vertices, 18 with the edges, 12 with the faces, 1 in the volume)
31	56-node fifth order tetrahedron (4 nodes associated with the vertices, 24 with the edges, 24 with the faces, 4 in the volume)
92	64-node third order hexahedron (8 nodes associated with the vertices, 24 with the edges, 24 with the faces, 8 in the volume)
93	125-node fourth order hexahedron (8 nodes associated with the vertices, 36 with the edges, 54 with the faces, 27 in the volume)

B Other functions

B.1 function fc_oogmsh.configure

This function configure the fc oogmsh Python toolbox to be used with **gmsh** by generating a configuration file. The name of this file is returned by the `fc_oogmsh.Sys.getLocalConfFile()` function.

Syntaxe

```
fc_oogmsh.configure()

fc_oogmsh.configure(Key=Value)
```

Description

`fc_oogmsh.configure()` uses default values to set:

- the location of the **gmsh** binary application,
- the directory of the *.geo* files used by **gmsh**,
- the directory where to save the *.msh* generated by **gmsh**.

`fc_oogmsh.configure(Key=Value)` specifies function options using one or more `Key`,`Value` pair arguments.
The `Key` options can be

- **gmsh** : to specify the location of the **gmsh** binary application,
- **geodir** : to specify the directory of the *.geo* files used by **gmsh**,
- **meshdir** : to specify the directory where to save the *.msh* generated by **gmsh**,
- **reset** : if `True` configure with default values (default : `False`)

B.2 function `fc_oogmsh.gmsh.elm_type_desc`

This function returns an array of dictionary which contains some informations on a `gmsh` *elt-type* described in Appendix A.

Syntaxe

```
elt=fc_oogmsh.gmsh.get_elm_desc_by_type(Type)
```

Python code with output

```
elt2=fc_oogmsh.gmsh.get_elm_desc_by_type(2)
print('elt2='+str(elt2))
elt4=fc_oogmsh.gmsh.get_elm_desc_by_type(4)
print('elt4='+str(elt4))
elt11=fc_oogmsh.gmsh.get_elm_desc_by_type(11)
print('elt11='+str(elt11))
```

```
elt2={'elm_type': 2, 'desc': '3-node triangle', 'nb_nodes': 3, 'order': 1, 'incomplete': False, 'd': 2, 'geo': 'triangle'}
elt4={'elm_type': 4, 'desc': '4-node tetrahedron', 'nb_nodes': 4, 'order': 1, 'incomplete': False, 'd': 3, 'geo': 'tetrahedron'}
elt11={'elm_type': 11, 'desc': '10-node second order tetrahedron (4 nodes associated with the vertices and 6 with the edges)', 'nb_nodes': 10, 'order': 2,
       'incomplete': False, 'd': 3, 'geo': 'tetrahedron'}
```

B.3 Matplotlib functions

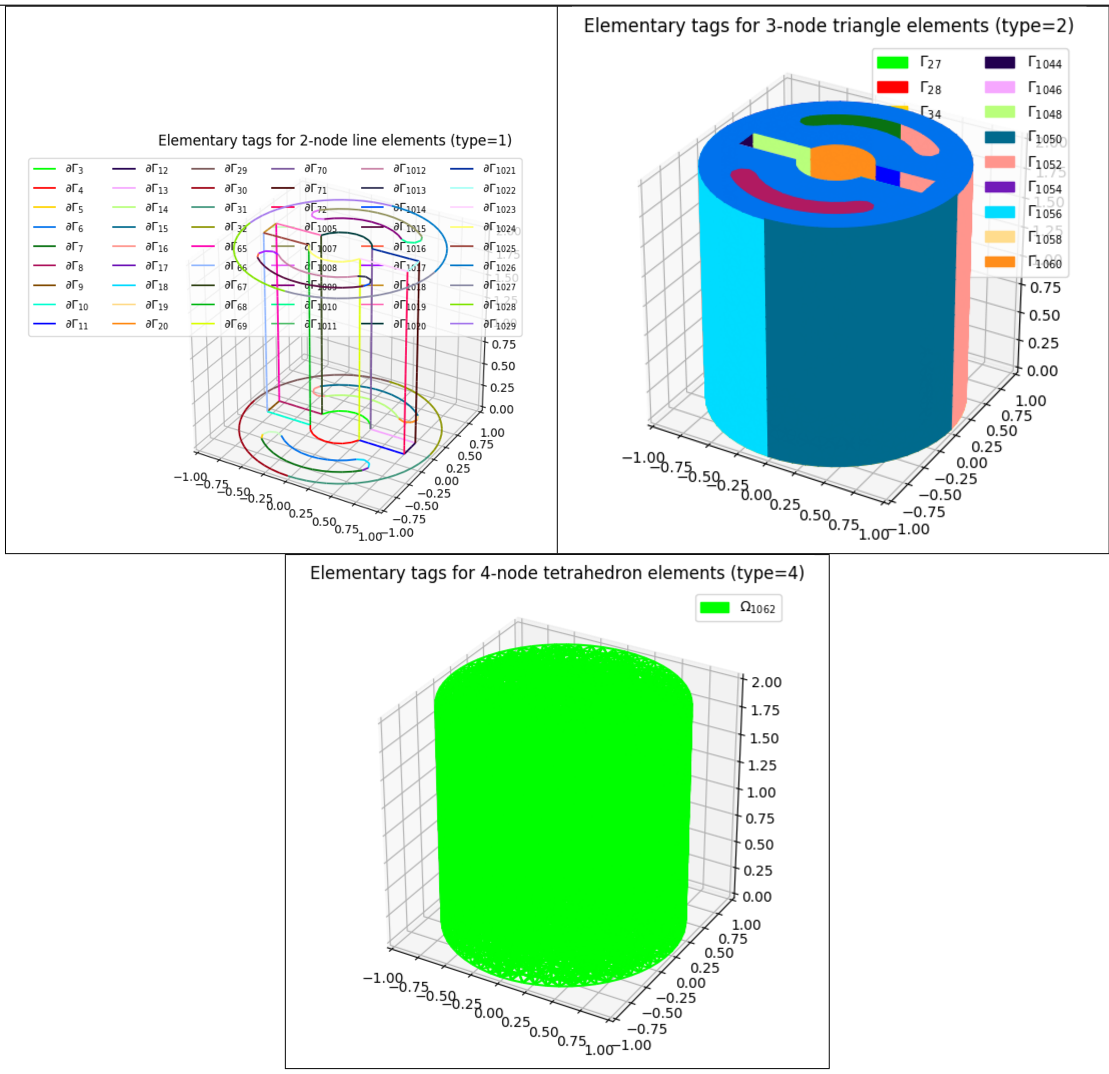
B.3.1 function `fc_oogmsh.matplotlib.plot_elementary_tags`

This function plot *Elementary Tags* of an `ooGmsh2` or `ooGmsh4` object of *Element Type*

- 1, *2-node line* elements,
- 2, *3-node triangle* elements,
- 4, *4-node tetrahedron* elements.

This function uses the `fc-matplotlib4mesh` package [1] version 0.1.0.

```
fc_oogmsh.Matplotlib.plot_elementary_tags(Gh)
```




```
meshfile=fc_oogmsh.gmsh.buildmesh3d('cylinderkey',10, verbose=0,force=True)
Gh = fc_oogmsh.ooGmsh4(meshfile)
fc_oogmsh.Matplotlib.plot_elementary_tags(Gh)
```

Listing 11: Using `fc_oogmsh.Matplotlib.plot_elementary_tags` function

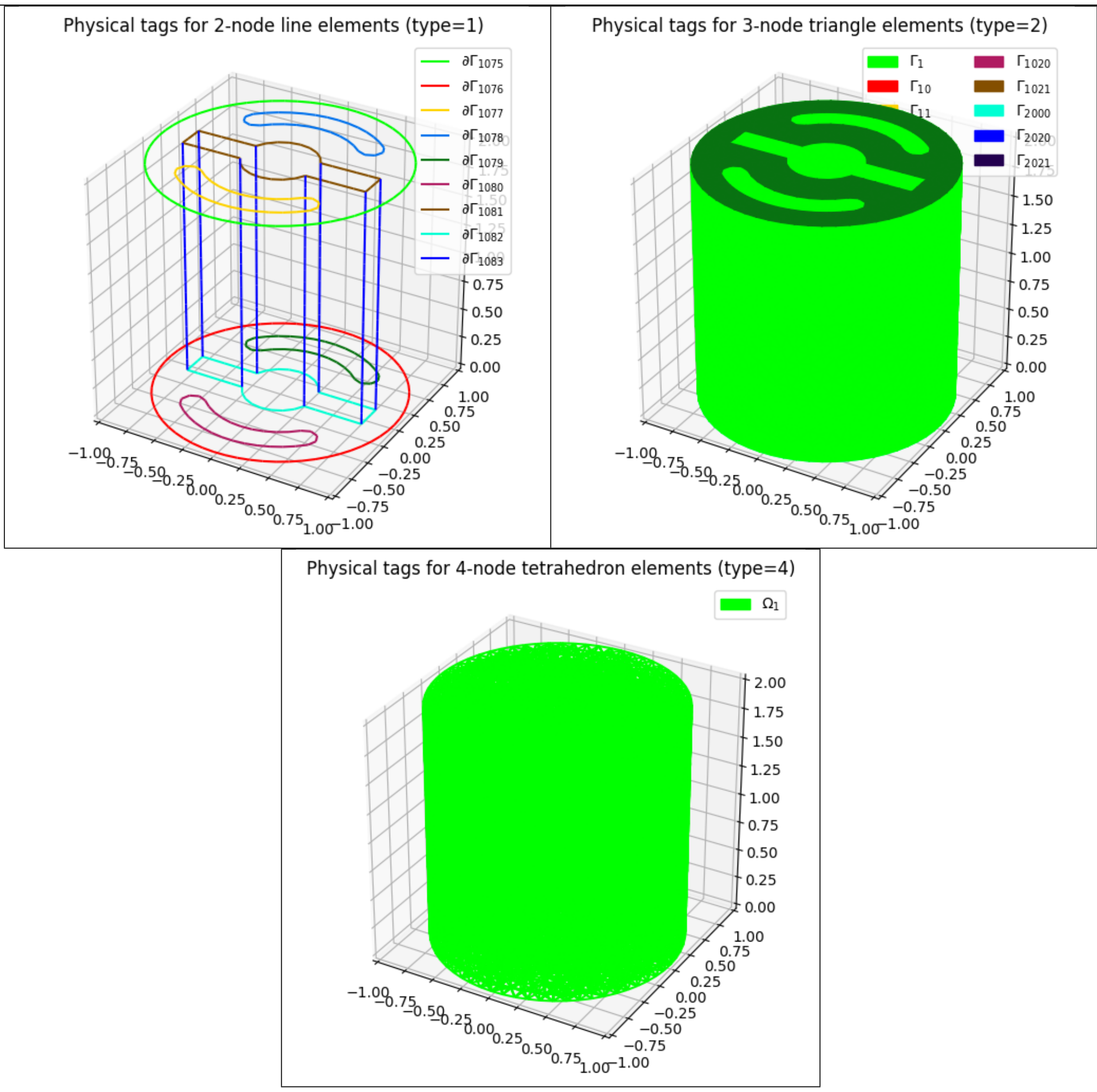
B.3.2 function `fc_oogmsh.matplotlib.plot_physical_tags`

This function plot *Elementary Tags* of an `ooGmsh2` or `ooGmsh4` object of *Element Type*

- 1, *2-node line* elements,
- 2, *3-node triangle* elements,
- 4, *4-node tetrahedron* elements.

This function uses the **fc-matplotlib4mesh** package [1] version 0.1.0.

```
fc_oogmsh.Matplotlib.plot_physical_tags(Gh)
```




```
meshfile=fc_oogmsh.gmsh.buildmesh3d('cylinderkey',10, verbose=0,force=True)
Gh = fc_oogmsh.ooGmsh4(meshfile)
fc_oogmsh.Matplotlib.plot_physical_tags(Gh)
```

Listing 12: Using `fc_oogmsh.Matplotlib.plot_physical_tags` function

B.4 Mayavi functions

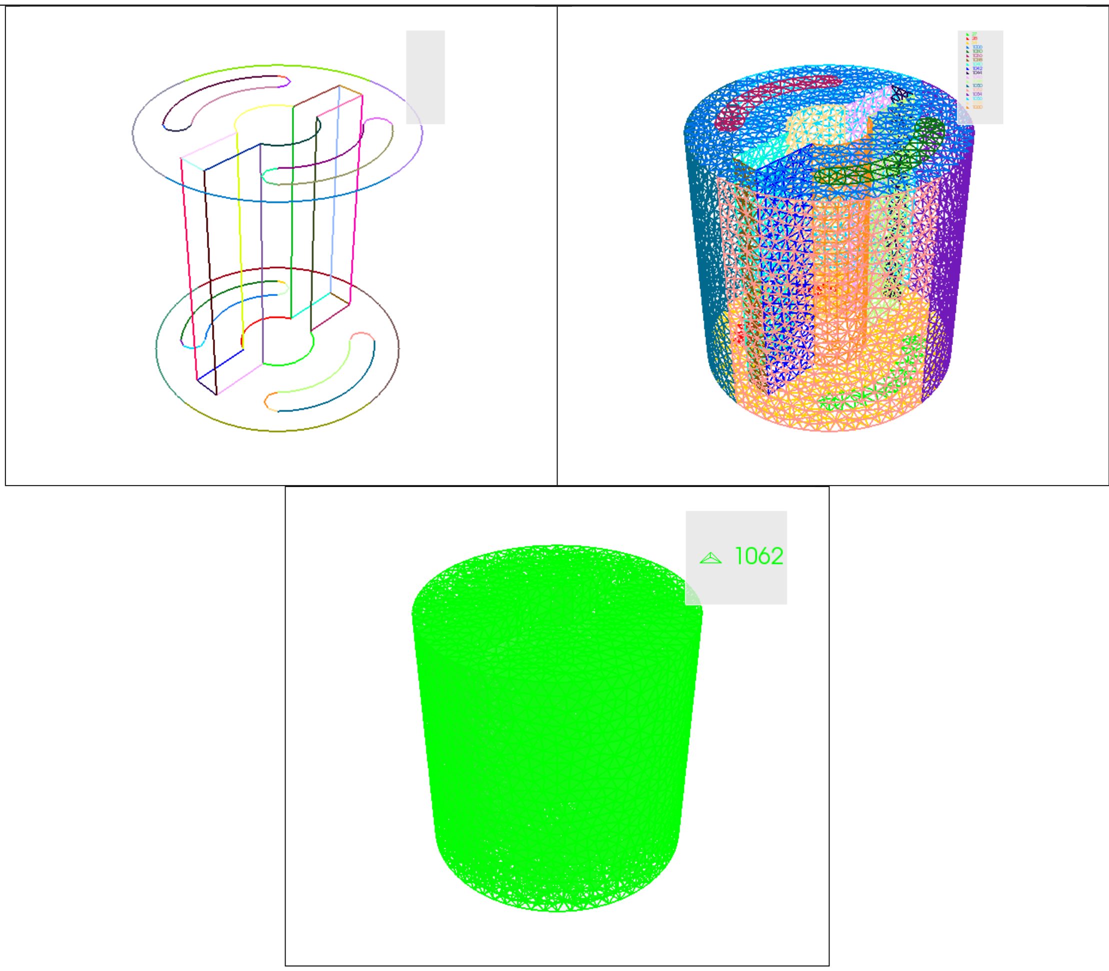
B.4.1 function `fc_oogmsh.mayavi.plot_elementary_tags`

This function plot *Elementary Tags* of an `ooGmsh2` or `ooGmsh4` object of *Element Type*

- 1, *2-node line* elements,
- 2, *3-node triangle* elements,
- 4, *4-node tetrahedron* elements.

This function uses the `fc-mayavi4mesh` package [2] version 0.1.0.

```
fc_oogmsh.Mayavi.plot_elementary_tags(Gh)
```




```
meshfile=fc_oogmsh.gmsh.buildmesh3d('cylinderkey',10, verbose=0,force=True)
Gh = fc_oogmsh.ooGmsh4(meshfile)
fc_oogmsh.Mayavi.plot_elementary_tags(Gh)
```

Listing 13: Using `fc_oogmsh.Mayavi.plot_elementary_tags` function

B References

[1] F. Cuvelier. fc_matplotlib4mesh: a Python package for displaying simplices meshes or datas on simplices meshes by using matplotlib python package. `http://www.math.univ-paris13.fr/~cuvelier/software/`, 2017. User's Guide.

[2] F. Cuvelier. fc_mayavi4mesh: a Python package for displaying simplices meshes or datas on simplices meshes by using mayavi python package. `http://www.math.univ-paris13.fr/~cuvelier/software/`, 2017. User's Guide.

[3] F. Cuvelier. fc_simesh_mayavi: an add-on to the fc_simesh Python package for displaying simplices meshes or datas on simplices meshes by using mayavi python package. `http://www.math.univ-paris13.fr/~cuvelier/software/`, 2017. User's Guide.

[4] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[5] C. Geuzaine and J.-F. Remacle. Gmsh 2.15.0. `http://gmsh.info`, 2016.

[6] C. Geuzaine and J.-F. Remacle. Gmsh 4.2.1. `http://gmsh.info`, 2019.

[7] Python.org. Python. `http://www.python.org/`, 2013.