meshtools Python package, User's Guide*

version 0.2.1

F. Cuvelier†

January 5, 2020

Abstract

The experimental fc-meshtools Python package contains some simpicial or *orthotope* meshes given by their vertices array q and connectivity array me where an *orthotope* mesh is tesselate with hexahedra in dimension 3 and quadrangle in dimension 2. Theses meshes can be easily used in other Python codes for debugging or testing purpose. With simplicial meshes, some functions are provided to compute, for each mesh elements, volumes, gradient of barycentric coordinates, ...

*LaTeX manual, revision 0.2.1, compiled with Python 3.8.1, and packages `fc-meshtools`[0.2.1], `fc-tools`[0.0.24],

†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

0 Contents

1 Introduction

A mesh is given by its vertices array q and its connectivity array me . For demonstration purpose, some meshes are given in this package and stored in the `fc_meshtools/data` directory. They can be load by using the function `getMesh` of the `fc_meshtools` module. This package contains two type of mesh : simplicial meshes (triangles in 2D and tetrahedra in 3D) and *orthotope* meshes (quadrangles in 2D and hexahedra in 3D).

Here are the kind of simplicial meshes present in this toolbox:

- a triangular mesh in dimension 2, made with 2-simplices (ie. triangles),
- a tetrahedral mesh in dimension 3, made with 3-simplices (ie. tetrahedron),
- a triangular mesh in dimension 3 (surface mesh), made with 2-simplices,
- a line mesh in dimension 2 or 3 made with 1-simplices (ie. lines).

Here are the kind of orthotope meshes present in this toolbox:

- a mesh in dimension 2 made with quadrangles,
- a mesh in dimension 3 made with hexahedra,
- a surface mesh in dimension 3, made with quadrangles,
- a line mesh in dimension 2 or 3 made with lines.

This toolbox was tested on various OS with Python releases (from python.org):

	Python				
Linux	**2.7.16**	**3.5.9**	**3.6.10**	**3.7.6**	**3.8.1**
CentOS 7.7.1908	✓	✓	✓	✓	✓
Debian 9.11	✓	✓	✓	✓	✓
Fedora 29	✓	✓	✓	✓	✓
OpenSUSE Leap 15.0	✓	✓	✓	✓	✓
Ubuntu 18.04.3 LTS	✓	✓	✓	✓	✓
Apple Mac OS X	**2.7.16**	**3.5.4**	**3.6.8**	**3.7.6**	**3.8.1**
MacOS High Sierra 10.13.6	✓	✓	✓	✓	✓
MacOS Mojave 10.14.4	✓	✓	✓	✓	✓
MacOS Catalina 10.15.2	✓	✓	✓	✓	✓
Microsoft Windows	**2.7.16**	**3.5.4**	**3.6.8**	**3.7.6**	**3.8.1**
Windows 10 (1909)	✓	✓	✓	✓	✓

Under all Linux distributions, Python releases are compiled from sources.

2 Installation

The fc meshtools Python package is available from the Python Package Index [2].

For Python 2, the installation of this package can be done with the `pip` or `pip2` command.

- For an installation which isolated to the current user, one can do:

```
$ pip install --user fc_meshtools
```

- For an installation for all users, one can do:

```
$ sudo pip install fc_meshtools
```

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

3 Meshes

This package contains two type of mesh : simplicial meshes (triangles in 2D and tetrahedra in 3D) and *orthotope* meshes (quadrangles in 2D and hexahedra in 3D).

3.1 Simplicial meshes

The functions `fc_meshtools.getMesh(dim,d)` returns a mesh vertices array `q`, a mesh elements connectivity array `me` and an indices array `toGlobal` associated with the input arguments `dim` (space dimension) and `d` (simplex dimension). The vertices array `q` is a `dim`-by-$\mathrm{n_q}$ array where `dim` is the space dimension (2 or 3) and $\mathrm{n_q}$ the number of vertices. The connectivity array `me` is a $(d+1)$-by-$\mathrm{n_{me}}$ array where $\mathrm{n_{me}}$ is the number of mesh elements and $0 \leqslant$ `d` $\leqslant$ `dim` is the simplicial dimension:

- `d` $= 0$: points,
- `d` $= 1$: lines,
- `d` $= 2$: triangle,
- `d` $= 3$: tetrahedron.

So we can use theses functions to obtain

- 3D mesh: `getMesh(3,3)` (*main* mesh), `getMesh(3,2)`, `getMesh(3,1)`, `getMesh(3,0)`,
- 3D surface mesh: `getMesh(3,2,surface=True)` (*main* mesh), `getMesh(3,1,surface=True)` , `getMesh(3,0,surface=True)`,
- 2D mesh: `getMesh(2,2)` (*main* mesh), `getMesh(2,1)`, `getMesh(2,0)`.

The indices array `toGlobal` contains the indices of the vertices in the *main* mesh

For example,

- `q3,me3,toGlobal3=fc_meshtools.getMesh(3,3)` return a 3-simplicial mesh (main mesh) in space dimension `dim` $= 3$,
- `q2,me2,toGlobal2=fc_meshtools.getMesh(3,2)` return a 2-simplicial mesh in space dimension `dim` $= 3$,
- `q1,me1,toGlobal1=fc_meshtools.getMesh(3,1)` return a 1-simplicial mesh in space dimension `dim` $= 3$.

The third output contains the indices of the vertices in the *main* mesh:

```
q3[:,toGlobal2] == q2
q3[:,toGlobal1] == q1
```

3.2 Orthotope meshes

The functions `fc_meshtools.getMesh(dim,d,type='orthotope')` returns a mesh vertices array `q`, a mesh elements connectivity array `me` and an indices array `toGlobal` associated with the input arguments `dim` (space dimension) and `d`. Each mesh element has 2^{d} vertices and the reference element is the unit `d`-orthotope. The vertices array `q` is a `dim`-by-$\mathrm{n_q}$ array where `dim` is the space dimension (2 or 3) and $\mathrm{n_q}$ the number of vertices. The connectivity array `me` is a 2^{d}-by-$\mathrm{n_{me}}$ array where $\mathrm{n_{me}}$ is the number of mesh elements and $0 \leqslant$ `d` $\leqslant$ `dim` is the orthotope dimension:

- `d` $= 0$: point,
- `d` $= 1$: line,
- `d` $= 2$: quadrangle,
- `d` $= 3$: hexahedron.

So we can use theses functions to obtain

- 3D mesh:

  `getMesh(3,3,type='orthotope')` (*main* mesh)

  and

  `getMesh(3,d,type='orthotope')` with `d` $\in [\![0,2]\!]$,

- 3D surface mesh:

  `getMesh(3,2,type='orthotope',surface=True)` (*main* mesh)

  and

  `getMesh(3,d,type='orthotope',surface=True)` with `d` $\in [\![0,1]\!]$,

- 2D mesh:

  `getMesh(2,2)` (*main* mesh)

  and

  `getMesh(2,d,type='orthotope')` with `d` $\in [\![0,1]\!]$.

The indices array `toGlobal` contains the indices of the vertices in the *main* mesh
For example,

- `q3,me3,toGlobal3=fc_meshtools.getMesh(3,3)` return a 3-simplicial mesh (main mesh) in space dimension $dim = 3$,
- `q2,me2,toGlobal2=fc_meshtools.getMesh(3,2)` return a 2-simplicial mesh in space dimension $dim = 3$,
- `q1,me1,toGlobal1=fc_meshtools.getMesh(3,1)` return a 1-simplicial mesh in space dimension $dim = 3$.

The third output contains the indices of the vertices in the *main* mesh:
`q3[:,toGlobal2] == q2`
`q3[:,toGlobal1] == q1`

4 Functions

4.1 getMesh functions

Returns a vertices array `q`, a connectivity array `me` and an indices array `toGlobal`.

Syntaxe

```
q,me,toGlobal=fc_meshtools.getMesh(dim,d)
q,me,toGlobal=fc_meshtools.getMesh(dim,d, key=value, ...)
```

Description

`q,me,toGlobal=fc_meshtools.getMesh(dim,d)`

Returns a `dim`-dimensional mesh with `d`-simplex elements given by a vertices array `q`, a connectivity array `me` and an indices array `toGlobal` depending on the value of the `d`.

For a 3D mesh, we have `dim` $= 3$ and `d` $\in [\![0, 3]\!]$. For a 2D mesh, we have `dim` $= 2$ and `d` $\in [\![0, 2]\!]$.

In Listing 1, some examples are provided to get a 3D mesh and some associated surfaces respectively tesselate with tetrahedra and triangles.

Listing 1: : examples of fc_meshtools.getMesh function usage

```
import fc_meshtools
q2,me2,toG2=fc_meshtools.getMesh(3,2)
print('q2: %s, me2: %s, toG2: %s'%(str(q2.shape),str(me2.shape),str(toG2.shape)))
q3,me3,toG3=fc_meshtools.getMesh(3,3)
print('q3: %s, me3: %s, toG3: %s'%(str(q3.shape),str(me3.shape),str(toG3.shape)))
print('Error: %.5e'%abs(q3[:,toG2]-q2).max())
```

Output

```
q2: (3, 6866), me2: (3, 13740), toG2: (6866,)
q3: (3, 17643), me3: (4, 88649), toG3: (17643,)
Error: 0.00000e+00
```

q,me,toGlobal=fc_meshtools.getMesh(dim,d, key=value, ...) specifies function options using one or more key,value pair arguments:

- **type** : to select type of mesh elements. Could be 'simplex' (default) or 'orthotope'.
- **surface** : to select a surface mesh if value is True (default is False). For a surface mesh, we have dim $= 3$ and d $\in [\![0, 2]\!]$.
- small : to select a very small mesh if value is True (default is False).
- verbose : to select verbose mode if value is True (default is False).

In Listing 2, some examples are provided to get a surface mesh and some associated curves respectively tessellate with quadrangles and lines.

Listing 2: : examples of fc_meshtools.getMesh function used to read a surface mesh with quadrangle elements

```
import fc_meshtools
q2,me2,toG2=fc_meshtools.getMesh(3,2,type='orthotope',surface=True,small=False,verbose=True)
print('q2: %s, me2: %s, toG2: %s'%(str(q2.shape),str(me2.shape),str(toG2.shape)))
q1,me1,toG1=fc_meshtools.getMesh(3,1,type='orthotope',surface=True,small=False,verbose=True)
print('q1: %s, me1: %s, toG1: %s'%(str(q1.shape),str(me1.shape),str(toG1.shape)))
print('Error: %.5e'%abs(q2[:,toG1]-q1).max())
```

Output

```
Reading file mesh2orth1order3Ds.npz
q2: (3, 2843), me2: (4, 2778), toG2: (2843,)
Reading file mesh1orth1order3Ds.npz
q1: (3, 253), me1: (2, 256), toG1: (253,)
Error: 0.00000e+00
```

4.2 Volumes function (simplicial mesh)

Syntaxe Returns all the element volumes of a mesh given by a vertices array `q` and a connectivity array `me`. One can refer to [1] for computationnal details.

Description

`vols=fc_meshtools.simplicial.Volumes(q,me)`

`vols[k]` is the volume of the `(k+1)`-th mesh element where its vertices are the columns of `q[me[:,k]]`.

In Listing 3, some examples are provided.

Listing 3: : examples of fc_meshtools.simplicial.Volumes function usage

```
from fc_meshtools.simplicial import getMesh3D,Volumes
q,me,toG=getMesh3D(2)
vols=Volumes(q,me)
print('q: %s, me: %s, vols: %s:'%(str(q.shape),str(me.shape),str(vols.shape)))
```

Output

```
q: (3, 6866), me: (3, 13740), vols: (13740,):
```

4.3 Gradient of barycentric coordinates (simplicial mesh)

Syntaxe Returns all the gradients of barycentric coordinates of each element of a simplicial mesh given by a vertices array q and a connectivity array me. One can refer to [1] for computationnal details.

Description

G=fc_meshtools.simplicial.GradBaCo(q,me)

G[k,:,i] is the gradient of the i-th barycentric coordinate of the k-th mesh element.

In Listing 4, some examples are provided.

Listing 4: : examples of fc_meshtools.simplicial.GradBaCo function usage

```
from fc_meshtools.simplicial import GradBaCo
from fc_meshtools import getMesh
q,me,toG=getMesh(3,3)
G=GradBaCo(q,me)
print('q: %s, me: %s, G: %s:'%(str(q.shape),str(me.shape),str(G.shape)))
```

Output

```
q: (3, 17643), me: (4, 88649), G: (88649, 3, 4):
```

4 References

[1] F. Cuvelier. Exact integration for products of power of barycentric coordinates over d-simplexes in R^n. http://hal.archives-ouvertes.fr/hal-00931066v1, June 2018. preprint.

[2] Python Software Foundation. Pypi, the python package index. https://pypi.python.org/, 2003–.

Informations for developpers/maintainers of the fc meshtools Python package

git informations on the packages used to build this manual

```
  name: fc-meshtools
   tag: 0.2.1
commit: ec087ad15b223964e1d6d068a5f8c7219763216e
  date: 2020-01-05
  time: 07-07-00
status: 0


  name: fc-tools
   tag: 0.0.24
commit: 2ae83c0d581962971179c005d0f88ab33286725c
  date: 2019-12-21
  time: 11-34-49
status: 0
```

git informations on the LaTeX package used to build this manual

```
  name: fctools
   tag:
commit: 7ad9c7de44262e116aa101aeae74c5e5aee6ef61
  date: 2019-10-30
  time: 13-57-21
status: True
```

git informations on the fc_config package used to build this distribution

```
  name: fc-config
   tag:
commit: c35551bcf6ad1d29b5079505cbd3886cd4bfe08d
  date: 2020-01-05
  time: 07-01-37
status: True
```