FC_MAYAVI4MESH package, User's Guide *

François Cuvelier†

April 22, 2017

Abstract

This Python package allows to display simplices meshes or datas on simplices meshes by using Mayavi.

Contents

*Compiled with Python 3.6.0, package FC_TOOLS-0.0.8, and the plotting librarie MAYAVI-4.5.0

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

This work was partially supported by ANR Dedales.

1 Introduction

The **experimental** package package use Mayavi (>= 4.5.0) for displaying simplicial meshes or datas on simplicial meshes. Simplicial meshes could be:

- 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).

A simplicial mesh is given by its vertices array q and its connectivity array me.

Installation of Mayavi package for Python 3.6.0 under the linux distributions CentOS 7 and Ubuntu 14.04 LTS is presented on my web page `http://www.math.univ-paris13.fr/~cuvelier` in the section *Informatique/Python*.

1.1 Prerequisite

to do

1.2 Installation

To do

1.3 Remarks

In all this report, for graphical representation, we use mlab4sim namespace for graphical functions using mayavi. There is the python code previously used before any of the listings given thereafter:

```
from mayavi import mlab
import fc_mayavi4mesh.simplicial_mayavi as mlab4sim
from fc_mayavi4mesh.demos import
    getMesh2D, getMesh3D, getMesh3Ds
```

The functions getMesh2D, getMesh3D and getMesh3Ds return a mesh vertices array q and a mesh elements connectivity array associated with the input argument d (simplex dimension). The vertices array q is a dim-by-n_q or n_q-by-dim numpy array where dim is the space dimension (2 or 3) and n_q the number of vertices. The connectivity array me is a $(d+1)$-by-n_{me} or n_{me}-by-$(d+1)$ numpy array where n_{me} is the number of mesh elements and $0 \leqslant d \leqslant dim$ is the simplicial dimension:

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

For example, (q,me)=getMesh3D(2) return a 2-simplicial mesh in space dimension $dim = 3$.

2 function PLOTMESH

The PLOTMESH function displays a mesh given by a vertices and connectivity arrays.

Syntaxe

```
mlab4sim.plotmesh(q,me)
mlab4sim.plotmesh(q,me,Key=Value, ...)
```

Description

mlab4sim.plotmesh(q,me,) displays all the d-dimensional simplices elements.

mlab4sim.plotmesh(q,me,Key=Value, ...) specifies function options using one or more Key,Value pair arguments. Options of first level are

- `color` : specify the mesh color as a RGB tupple or a string color (see check_color ...)
- `cute_planes` : cut mesh by n plans given by a list of Plane objects (only in dimension 3). The Plane constructor is Plane(origin=[x,y,z], normal=[nx,ny,z]) which defined the plane coming through point origin and orthogonal to the vector normal. The normal vector pointed to the part of the mesh not displayed. (only in dimension 3) default : [] (no cut).

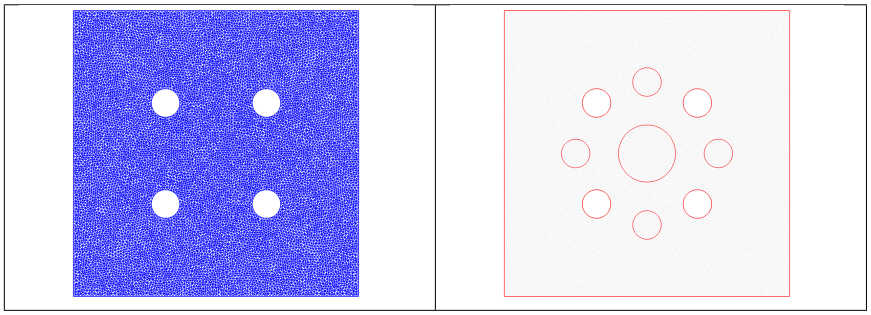
The options of second level depend on the type of elementaries mesh elements to represent.
One can use any option of the following functions according to the type of d-simplex to be represented.

- In dimension 3,
  - if $d == 3$, mlab.pipeline. surface function is used with tvtk.UnstructuredGrid and tvtk.Tetra().cell_type

- if $d == 2$, mlab.pipeline.surface function is used with tvtk.UnstructuredGrid and tvtk.Triangle().cell_type
- if $d == 1$, mlab.pipeline.surface function is used with tvtk.UnstructuredGrid and tvtk.Line().cell_type
- if $d == 0$, not yet implemented

- In dimension 2,
  - if $d == 2$, mlab.triangular_mesh function is used,
  - if $d == 1$, if $d == 1$, mlab.pipeline.surface function is used with tvtk.UnstructuredGrid and tvtk.Line().cell_type
  - if $d == 0$, not yet implemented
- dimension 1, not yet implemented

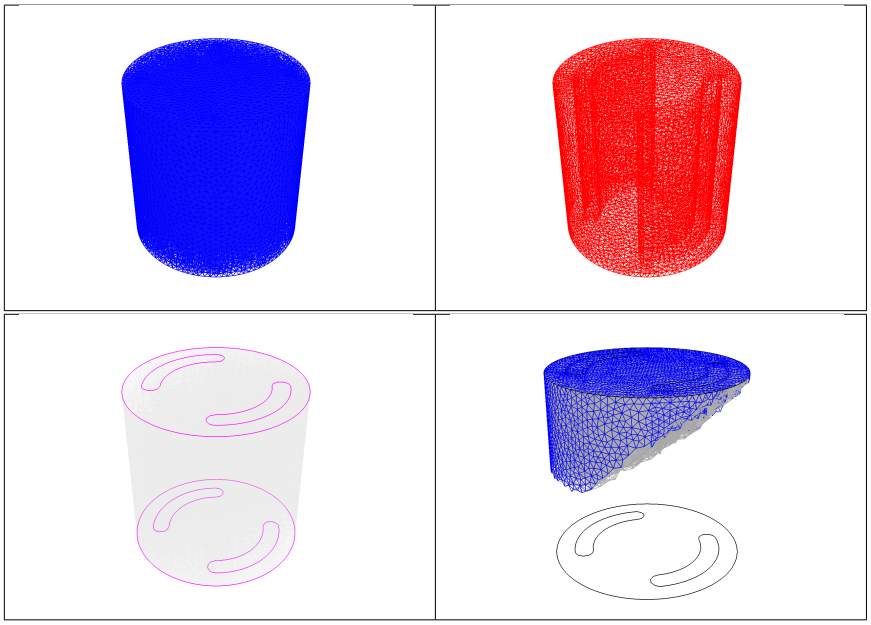
2.1 2D example



```
q2,me2=getMesh2D(2)
q1,me1=getMesh2D(1)
mlab.close(all=True)
mlab.figure(1)
mlab4sim.plotmesh(q2,me2)
mlab.view(0,0)
mlab.figure(2)
mlab4sim.plotmesh(q1,me1,color='Red',line_width=2)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.1)
mlab.view(0,0)
```

Listing 1: 2D plot mesh

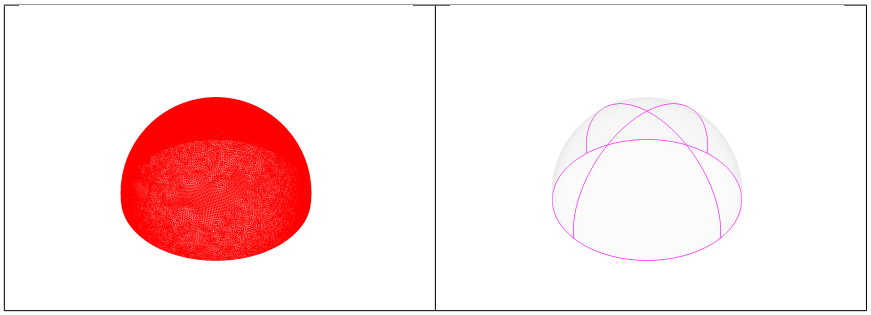
2.2 3D example



```
q3,me3=getMesh3D(3)
q2,me2=getMesh3D(2)
q1,me1=getMesh3D(1)
mlab.close(all=True)
mlab.figure(1)
mlab4sim.plotmesh(q3,me3)
mlab.figure(2)
mlab4sim.plotmesh(q2,me2,color='Red')
mlab.figure(3)
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.02)
mlab4sim.plotmesh(q1,me1,color='Magenta',line_width=2)
from fc_tools.graphics import Plane
P=[Plane(origin=[0,0,1],normal=[0,0,-1]), Plane(origin=[0,0,1],normal=[0,-1,-1])]
mlab.figure(4)
mlab4sim.plotmesh(q3,me3,cut_planes=P,color='DarkGrey')
mlab4sim.plotmesh(q2,me2,cut_planes=P)
mlab4sim.plotmesh(q1,me1,color='Black',line_width=2)
mlab.view(-146,67,6)
```

Listing 2: 3D plot mesh

2.3 3D surface example



```
q2,me2=getMesh3Ds(2)
q1,me1=getMesh3Ds(1)
mlab.close(all=True)
mlab.figure(1)
mlab4sim.plotmesh(q2,me2,color='Red')
mlab.figure(2)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.02)
mlab4sim.plotmesh(q1,me1,color='Magenta',line_width=2)
```

Listing 3: 3D surface mesh : plot function

3 function PLOT

The PLOT function displays scalar datas on a mesh given by vertices and connectivity arrays.

Syntaxe

```
mlab4sim.plot(q,me,u)
mlab4sim.plot(q,me,u,Key=Value, ...)
```

Description

`mlab4sim.plot(q,me,u)` displays data u on the given mesh. The data u is an 1D-array of size n_q

`mlab4sim.plot(q,me,u,Key=Value, ...)` specifies function options using one or more Key,Value pair arguments. Options of first level are

- `plane` : if True, (default : False)

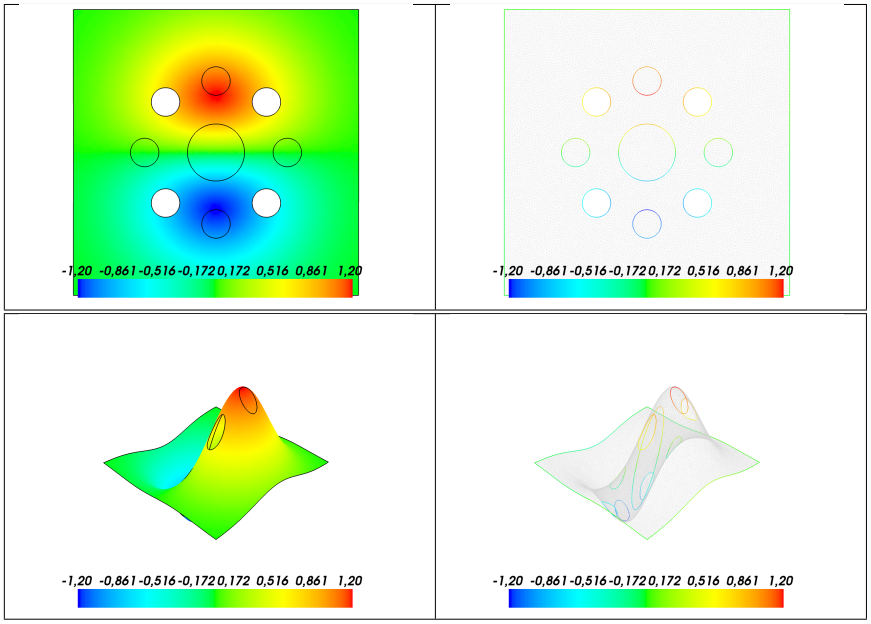
The second level options depend on the type of elementaries mesh elements on which we want to represent datas.
One can use any option of the following functions according to the type of d-simplex.

- In dimension 3,
  - if $d == 3$, mlab.pipeline. surface function is used with tvtk.UnstructuredGrid and tvtk.Tetra().cell_type

- if $d == 2$, mlab.triangular_mesh function is used.
- if $d == 1$, mlab.pipeline. surface function is used with tvtk.UnstructuredGrid and tvtk.Line().cell_type

- In dimension 2,
  - if $d == 2$, mlab.triangular_mesh function is used.
  - if $d == 1$, mlab.pipeline. surface function is used with tvtk.UnstructuredGrid and tvtk.Line().cell_type.
- Dimension 1 : not implemented.

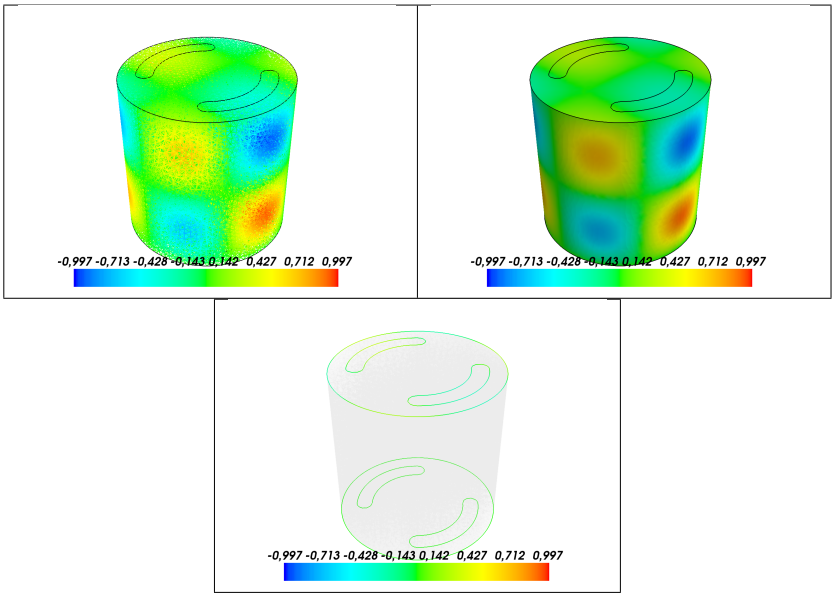
3.1 2D example




```
q2,me2=getMesh2D(2)
q1,me1=getMesh2D(1)
f=lambda x,y: 5*np.exp(-3*(x**2+y**2))*np.cos(x)*np.sin(y)
u2=f(q2[:,0],q2[:,1])
u1=f(q1[:,0],q1[:,1])
mlab.close(all=True)
mlab.figure(1)
mlab4sim.plot(q2,me2,u2)
mlab4sim.plotmesh(q1,me1,color='Black',line_width=2)
mlab.colorbar()
mlab.view(0,0)
mlab.figure(2)
mlab4sim.plot(q1,me1,u1,line_width=2)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.1)
mlab.view(0,0)
mlab.colorbar()
mlab.figure(3)
mlab4sim.plot(q2,me2,u2,plane=False)
mlab4sim.plotmesh(q1,me1,z=u1,color='Black',line_width=2)
mlab.colorbar()
mlab.figure(4)
mlab4sim.plot(q1,me1,u1,line_width=2,plane=False)
mlab4sim.plotmesh(q2,me2,z=u2,color='LightGray',opacity=0.1)
mlab.colorbar()
```

Listing 4: 2D mesh : plot function

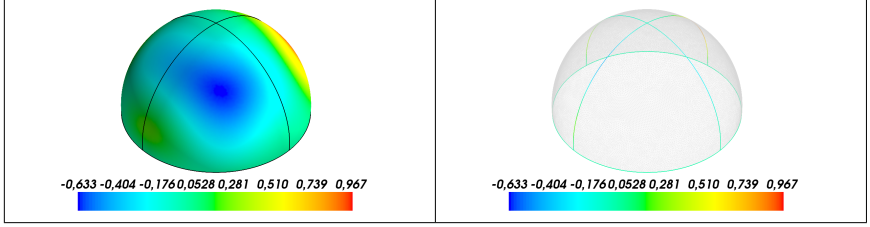
3.2 3D example




```
q3,me3=getMesh3D(3)
q2,me2=getMesh3D(2)
q1,me1=getMesh3D(1)
f=lambda x,y,z: np.cos(3*x)*np.sin(2*y)*np.sin(3*z)
u3=f(q3[:,0],q3[:,1],q3[:,2])
u2=f(q2[:,0],q2[:,1],q2[:,2])
u1=f(q1[:,0],q1[:,1],q1[:,2])
mlab.close(all=True)
mlab.figure(1)
mlab4sim.plot(q3,me3,u3)
mlab4sim.plotmesh(q1,me1,color='Black',line_width=2)
mlab.colorbar()
mlab.figure(2)
mlab4sim.plot(q2,me2,u2)
mlab4sim.plotmesh(q1,me1,color='Black',line_width=2)
mlab.colorbar()
mlab.figure(3)
mlab4sim.plot(q1,me1,u1,line_width=2,vmin=min(u3),vmax=max(u3))
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.02)
mlab.colorbar()
```

Listing 5: 3D mesh : plot function

3.3 3D surface example




```
q2,me2=getMesh3Ds(2)
q1,me1=getMesh3Ds(1)
f=lambda x,y,z: np.cos(3*x-1)*np.sin(2*y-2)*np.sin(3*z)
u2=f(q2[:,0],q2[:,1],q2[:,2])
u1=f(q1[:,0],q1[:,1],q1[:,2])
mlab.close(all=True)
mlab.figure(1)
mlab4sim.plot(q2,me2,u2)
mlab4sim.plotmesh(q1,me1,color='Black',line_width=2)
mlab.colorbar()
mlab.figure(2)
mlab4sim.plot(q1,me1,u1,line_width=2,vmin=min(u2),vmax=max(u2))
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.1)
mlab.colorbar()
```

Listing 6: 3D surface mesh : plot function

4 function PLOTISO

The PLOTISO function displays isolines from datas on a mesh given by a vertices and connectivity arrays. This function only works with 2-simplices in space dimension 2 or 3.

Syntaxe

```
mlab4sim.plotiso(q,me,u)
mlab4sim.plotiso(q,me,u,Key=Value, ...)
```

Description

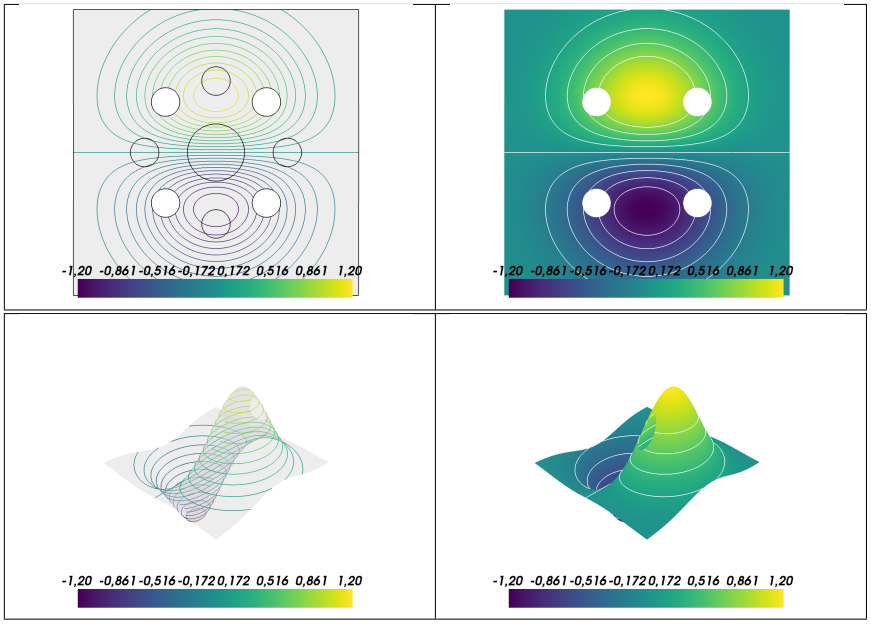
mlab4sim.plotiso(q,me,u) displays data u on all the 2-dimensional simplices elements as colored isovalues. The data u is an 1D-array of size nq.

mlab4sim.plotiso(q,me,u,key=value, ...) specifies function options using one or more key,value pair arguments. Options of first level are

- `contours` : to specify the number of isolines (default : 10) or a list/numpy array of isovalues (default : empty)
- `plane` : if False draw 3D isovalues with data u as z values. (default : True)
- `color` : to specify one color for all isolines (default : None))

The second level options are the options of the mlab.pipeline.iso_surface which we use to draw the isovalues.

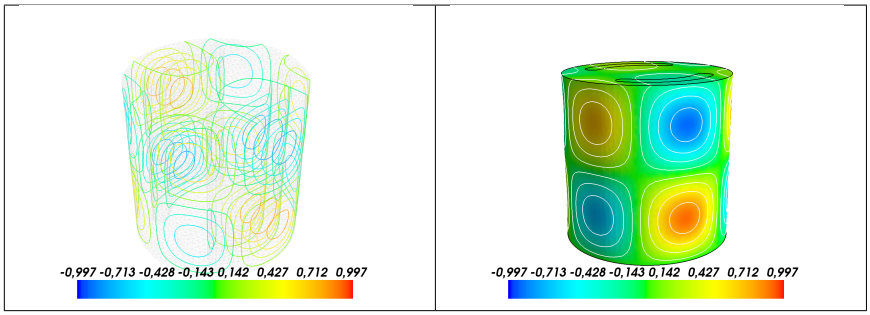
4.1 2D example




```
q2,me2=getMesh2D(2)
q1,me1=getMesh2D(1)
f=lambda x,y: 5*np.exp(-3*(x**2+y**2))*np.cos(x)*np.sin(y)
u2=f(q2[:,0],q2[:,1])
mlab.close(all=True)
mlab.figure(1)
mlab4sim.plotiso(q2,me2,u2,contours=25,colormap='viridis')
mlab4sim.plotmesh(q1,me1,color='black')
mlab4sim.plotmesh(q2,me2,color='LightGray',representation='surface',opacity=0.4)
mlab.view(0,0)
mlab.colorbar()
mlab.figure(2)
mlab4sim.plot(q2,me2,u2,colormap='viridis')
mlab4sim.plotiso(q2,me2,u2,contours=np.arange(-1,1,0.2),colormap='viridis',color='White')
mlab.view(0,0)
mlab.colorbar()
mlab.figure(3)
mlab4sim.plotiso(q2,me2,u2,contours=25,colormap='viridis',plane=False)
#mlab4sim.plotmesh(q1,me1,color='black',plane=False)
mlab4sim.plotmesh(q2,me2,color='LightGray',representation='surface',opacity=0.4,z=u2)
mlab.colorbar()
mlab.figure(4)
mlab4sim.plot(q2,me2,u2,colormap='viridis',plane=False)
mlab4sim.plotiso(q2,me2,u2,contours=np.arange(-1,1,0.2),colormap='viridis',color='White',plane=False)
mlab.colorbar()
```

Listing 7: 2D mesh : plotiso function

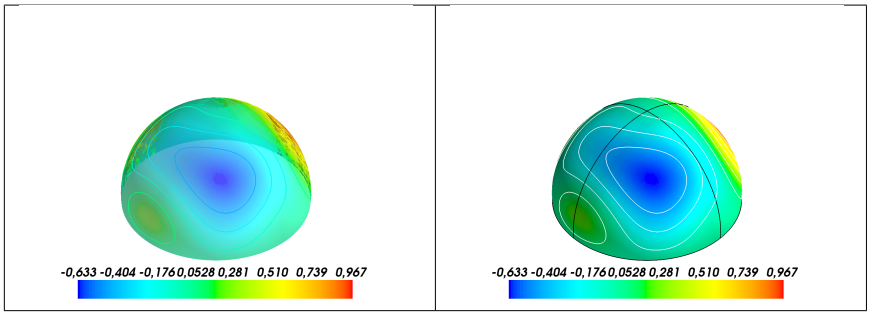
4.2 3D example




```
q3,me3=getMesh3D(3)
q2,me2=getMesh3D(2)
q1,me1=getMesh3D(1)
f=lambda x,y,z: np.cos(3*x)*np.sin(2*y)*np.sin(3*z)
u3=f(q3[:,0],q3[:,1],q3[:,2])
u2=f(q2[:,0],q2[:,1],q2[:,2])
u1=f(q1[:,0],q1[:,1],q1[:,2])
mlab.close(all=True)
mlab.figure(1)
mlab4sim.plotiso(q2,me2,u2,line_width=2)
mlab4sim.plotmesh(q2,me2, color='LightGray', opacity=0.1)
mlab.colorbar()
mlab.figure(2)
mlab4sim.plot(q2,me2,u2)
mlab4sim.plotiso(q2,me2,u2,line_width=2,color='White')
mlab4sim.plotmesh(q1,me1, color='Black')
mlab.colorbar()
mlab.view(65,74,7)
```

Listing 8: 3D mesh : plotiso function

4.3 3D surface example




```
q2,me2=getMesh3Ds(2)
q1,me1=getMesh3Ds(1)
f=lambda x,y,z: np.cos(3*x-1)*np.sin(2*y-2)*np.sin(3*z)
u2=f(q2[:,0],q2[:,1],q2[:,2])
u1=f(q1[:,0],q1[:,1],q1[:,2])
mlab.close(all=True)
mlab.figure(1)
mlab4sim.plot(q2,me2,u2, opacity=0.7)
mlab4sim.plotiso(q2,me2,u2, line_width=1.5)
mlab.colorbar()
mlab.figure(2)
mlab4sim.plot(q2,me2,u2)
mlab4sim.plotiso(q2,me2,u2, line_width=1.5,color='White')
mlab4sim.plotmesh(q1,me1, color='Black')
mlab.colorbar()
```

Listing 9: 3D surface mesh : plotiso function

5 function QUIVER

The QUIVER function displays vector field datas on a mesh given by a vertices and connectivity arrays.

Syntaxe

```
mlab4sim.quiver(q,me,V)
mlab4sim.quiver(q,me,V,Key=Value, ...)
```

Description

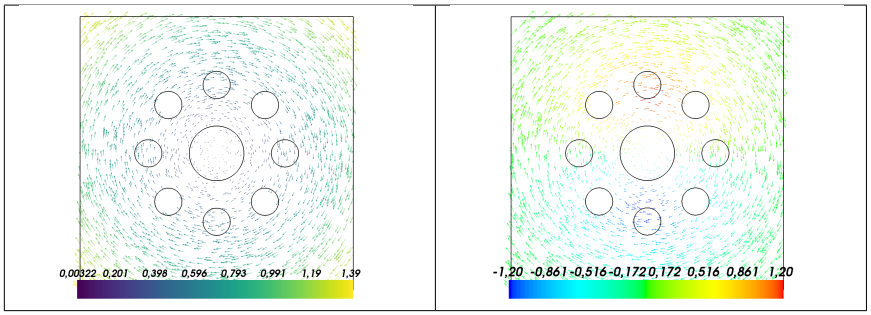
mlab4sim.quiver(q,me,V) displays vector field V on each vertices of the d-dimensional simplices elements in dimension $d = 2$ or $d = 3$. The data V is an 2D-array numpy array of size *dim*-by-nq .

mlab4sim.quiver(q,me,V,Key=Value, ...) specifies function options using one or more Key,Value pair arguments. Options of first level are

- `scalars` : to set quivers color to a numpy array of size nq (default : empty and use colors of the mesh elements).
- `color` : to specify one color for all quivers.

For Key/Value pairs, one could also used those of the mlab.quiver3d function.

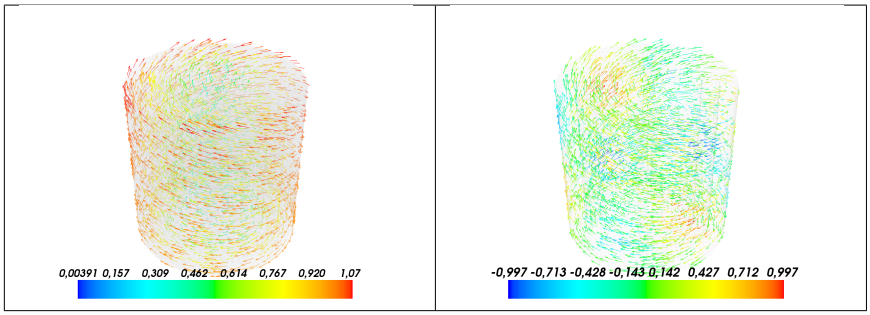
5.1 2D example




```
q2,me2=getMesh2D(2)
q1,me1=getMesh2D(1)
f=lambda x,y: 5*np.exp(-3*(x**2+y**2))*np.cos(x)*np.sin(y)
u=f(q2[:,0],q2[:,1])
w=[lambda x,y: y*np.cos(-(x**2+y**2)/10), lambda x,y: -x*np.cos(-(x**2+y**2)/10)]
W=np.array([w[0](q2[:,0],q2[:,1]),w[1](q2[:,0],q2[:,1])])
mlab.close(all=True)
nq=q2.shape[0]
mlab.figure(1)
mlab4sim.quiver(q2,me2,W, scale_factor=0.05 ,line_width=1,
    mask_points=nq//2000,colormap='viridis')
mlab4sim.plotmesh(q1,me1,color='black')
mlab.vectorbar()
mlab.view(0,0)
mlab.figure(2)
mlab4sim.quiver(q2,me2,W, scalars=u, scale_factor=0.05,line_width=1,
    mask_points=nq//2000)
mlab4sim.plotmesh(q1,me1,color='black')
mlab.scalarbar()
mlab.view(0,0)
```

Listing 10: 2D mesh : quiver function

5.2 3D example

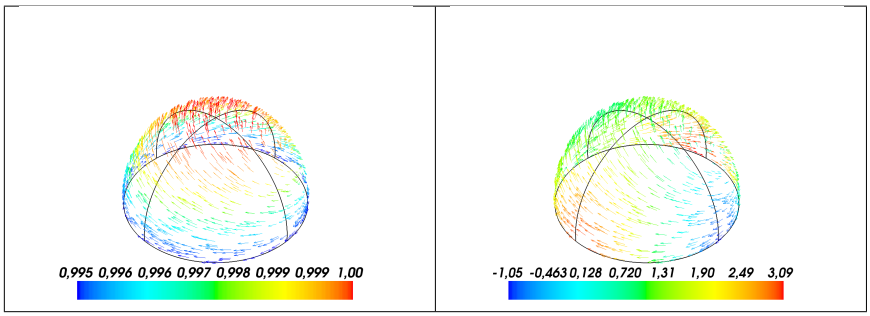



```
q3,me3=getMesh3D(3)
q2,me2=getMesh3D(2)
q1,me1=getMesh3D(1)
f=lambda x,y,z: np.cos(3*x)*np.sin(2*y)*np.sin(3*z)
u=f(q3[:,0],q3[:,1],q3[:,2])
w=[lambda x,y,z: y*np.cos(-(x**2+y**2)/10), lambda x,y,z:
    -x*np.cos(-(x**2+y**2)/10), lambda x,y,z: z/5]
W=np.array([w[0](q3[:,0],q3[:,1],q3[:,2]),
            w[1](q3[:,0],q3[:,1],q3[:,2]),
            w[2](q3[:,0],q3[:,1],q3[:,2])])
mlab.close(all=True)
nq=q3.shape[0]
mlab.figure(1)
mlab4sim.quiver(q3,me3,W,line_width=1, mask_points=nq//3000)
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.05)
mlab.vectorbar()
mlab.figure(2)
mlab4sim.quiver(q3,me3,W,scalars=u, line_width=1, mask_points=nq//3000)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
mlab.scalarbar()
```

Listing 11: 3D mesh : quiver function

5.3 3D surface example

The following example use the *.geo* file `demisphere5.geo` which is in the directory `geodir` of the toolbox. This file contains description of a 3D surface mesh with simplices of dimensions 1 and 2.




```
q2,me2=getMesh3Ds(2)
q1,me1=getMesh3Ds(1)
f=lambda x,y,z: 3*x**2-y**3+z**2+x*y
u=f(q2[:,0],q2[:,1],q2[:,2])
w=[lambda x,y,z: y*np.cos(-(x**2+y**2)/10), lambda x,y,z:
    -x*np.cos(-(x**2+y**2)/10), lambda x,y,z: z]
W=np.array([w[0](q2[:,0],q2[:,1],q2[:,2]),
            w[1](q2[:,0],q2[:,1],q2[:,2]),
            w[2](q2[:,0],q2[:,1],q2[:,2])])
mlab.close(all=True)
nq=q2.shape[0]
mlab.figure(1)
mlab4sim.quiver(q2,me2,W,line_width=1,mask_points=nq//1000)
mlab4sim.plotmesh(q1,me1,color='black',line_width=2)
mlab.vectorbar()
mlab.figure(2)
mlab4sim.quiver(q2,me2,W,scalars=u, line_width=1,mask_points=nq//1000)
mlab4sim.plotmesh(q1,me1,color='black',line_width=2)
mlab.scalarbar()
```

Listing 12: 3D surface mesh : quiver function

6 function SLICEMESH

The SLICEMESH function displays intersection of a plane and a 3D mesh given by vertices and connectivity arrays.

Syntaxe

```
mlab4sim.slicemesh(q,me)
mlab4sim.slicemesh(q,me,Key=Value, ...)
```

Description

mlab4sim.slicemesh(q,me,) displays intersection of a plane and all the 3-dimensional simplices elements. By default the plane is given by an *origin* point $(0,0,0)$ which lies on it and a *normal* vector $(0,0,1)$ which is orthogonal to the plane.

mlab4sim.slicemesh(q,me,Key=Value, ...) specifies function options using one or more Key,Value pair arguments. Key could be:

- `origin` : to specify a point lying on the plane (default is $(0,0,0)$)
- `normal` : to specify a vector orthogonal to the plane (default is $(0,0,1)$)

The other Key/Value pair options are those of scalar_cut_plane function from mayavi.mlab.pipeline.

```
q3,me3=getMesh3D(3)
q2,me2=getMesh3D(2)
q1,me1=getMesh3D(1)
mlab.close(all=True)
mlab.figure(1)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
mlab4sim.slicemesh(q3,me3,origin=(0,0,1),normal=(-1,0,1))
mlab.view(132,53,7)
mlab.figure(2)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
for normal,color in [((1,0,0),'red'),((0,1,0),'magenta'),((0,0,1),'Maroon')]:
  mlab4sim.slicemesh(q3,me3,origin=(0,0,1),normal=normal,color=color)
mlab.view(155,66,7)
```

Listing 13: 3D mesh : slicemesh function

7 function SLICE

The SLICE function displays datas on the intersection of a plane and a 3D mesh given by vertices and connectivity arrays.

Syntaxe

```
mlab4sim.slice(q,me,u)
mlab4sim.slice(q,me,u,Key=Value, ...)
```

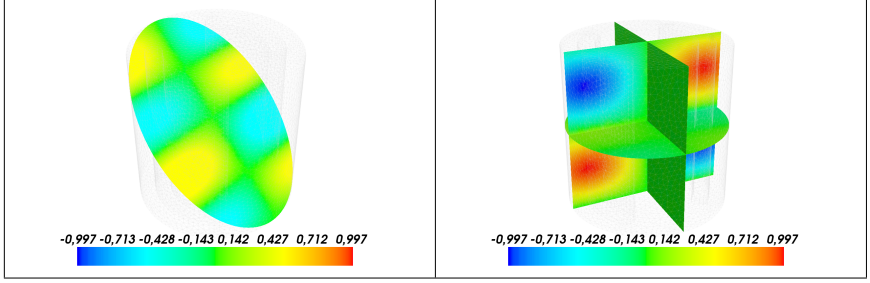
Description

mlab4sim.**slice**(q,me,u) displays u data on the intersection of a plane and all the 3-dimensional simplices elements. By default the plane is given by an *origin* point $(0,0,0)$ which lies on it and a *normal* vector $(0,0,1)$ which is orthogonal to the plane. The data u is an 1D-array of size nq.

mlab4sim.**slice**(q,me,u,Key,Value, ...) specifies function options using one or more Key,Value pair arguments. Key could be:

- `origin` : to specify a point lying on the plane (default is $(0,0,0)$)
- `normal` : to specify a vector orthogonal to the plane (default is $(0,0,1)$)

The other Key/Value pair options are those of scalar_cut_plane function from mayavi.mlab.pipeline.




```
q3,me3=getMesh3D(3)
q2,me2=getMesh3D(2)
q1,me1=getMesh3D(1)
f=lambda x,y,z: np.cos(3*x)*np.sin(2*y)*np.sin(3*z)
u3=f(q3[:,0],q3[:,1],q3[:,2])
u2=f(q2[:,0],q2[:,1],q2[:,2])
u1=f(q1[:,0],q1[:,1],q1[:,2])
mlab.close(all=True)
mlab.figure(1)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
mlab4sim.slice(q3,me3,u3,origin=(0,0,1),normal=(-1,0,1))
mlab.view(132,53,7)
mlab.colorbar()
mlab.figure(2)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
for normal in [(1,0,0),(0,1,0),(0,0,1)]:
  mlab4sim.slice(q3,me3,u3,origin=(0,0,1),normal=normal)
mlab.view(155,66,7)
mlab.colorbar()
```

Listing 14: 3D mesh : **slice** function

8 function SLICEISO

The SLICEISO function displays isolines of data on the intersection of a plane and a 3D mesh given by vertices and connectivity arrays.

Syntaxe

```
mlab4sim.sliceiso(q,me,u,P)
mlab4sim.sliceiso(q,me,u,Key=Value, ...)
```

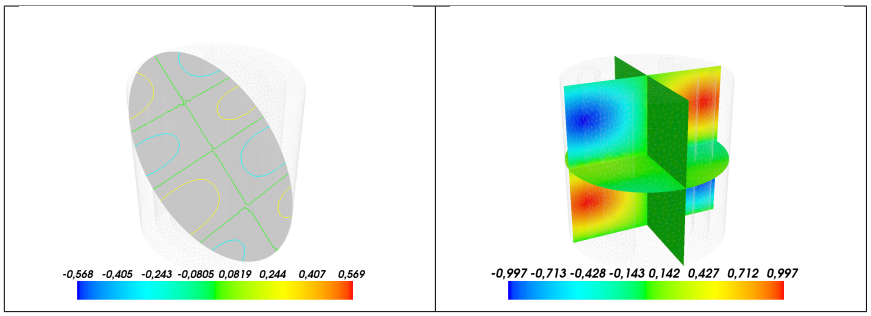
Description

`mlab4sim.sliceiso(q,me,u)` displays u data as isolines on the intersection of a plane and all the 3-dimensional simplices elements. By default the plane is given by an *origin* point $(0,0,0)$ which lies on it and a *normal* vector $(0,0,1)$ which is orthogonal to the plane.

`mlab4sim.sliceiso(q,me,u,key=value, ...)` allows additional key/value pairs to be used when displaying u. The key could be

- `contours` : to specify the number of isolines (default : 10) or a list/numpy array of isovalues (default : empty)
- `color` : to specify one color for all isolines (default : empty)

For key/value pairs, one could also used those of the iso_surface function from mayavi.mlab.pipeline.




```
q3,me3=getMesh3D(3)
q2,me2=getMesh3D(2)
q1,me1=getMesh3D(1)
f=lambda x,y,z: np.cos(3*x)*np.sin(2*y)*np.sin(3*z)
u3=f(q3[:,0],q3[:,1],q3[:,2])
u2=f(q2[:,0],q2[:,1],q2[:,2])
u1=f(q1[:,0],q1[:,1],q1[:,2])
mlab.close(all=True)
mlab.figure(1)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
mlab4sim.sliceiso(q3,me3,u3,origin=(0,0,1),normal=(-1,0,1))
mlab4sim.slicemesh(q3,me3,origin=(0,0,1),normal=(-1,0,1),color='DarkGray')
mlab.view(132,53,7)
mlab.colorbar()
mlab.figure(2)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
for normal in [(1,0,0),(0,1,0),(0,0,1)]:
    mlab4sim.slicemesh(q3,me3,origin=(0,0,1),normal=normal, color='LightGray')
    mlab4sim.slice(q3,me3,u3,origin=(0,0,1),normal=normal)
mlab.view(155,66,7)
mlab.colorbar()
```

Listing 15: 3D mesh : sliceiso function

9 function STREAMLINE

The STREAMLINE function allows to draw streamlines for given vector data and colorized by scalar data on a 3D mesh given by vertices and connectivity arrays. This supports various types of seed objects (line, sphere, plane and point seeds). It also allows to draw ribbons or tubes and further supports different types of interactive modes of calculating the streamlines.

Syntaxe

```
mlab4sim.streamline(q,me,u,V)
mlab4sim.streamline(q,me,u,V,Key=Value, ...)
```

Description

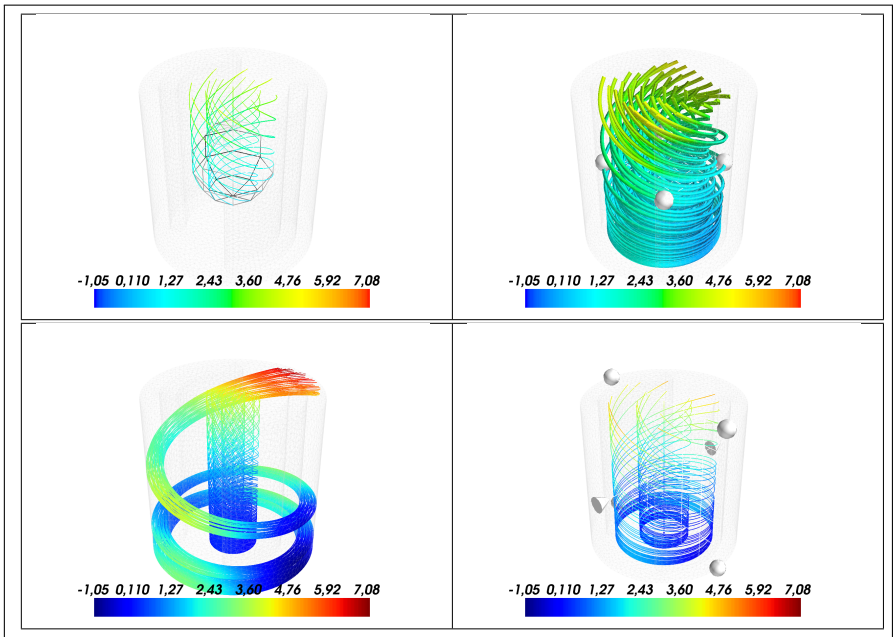
mlab4sim.streamline(q,me,u,V) displays streamlines computed from the vector data V and colorized by the scalar data u

mlab4sim.streamline(q,me,u,V,Key=Value, ...) specifies function options using one or more Key,Value pair arguments. Key could be:

- `labels` : to select the labels of the siMeshElt on which to draw the streamlines.
- `seed_options` :
- `seed_widget_options` :
- `streamtracer_options` :

The other Key/Value pair options are those of streamline function from mayavi.mlab.pipeline.

The following example use the *.geo* file `cylinder3dom.geo` which is in the directory `geodir` of the toolbox. This file contains description of a 3D mesh with simplices of dimensions 1, 2 and 3.




```
q3,me3=getMesh3D(3)
q2,me2=getMesh3D(2)
q1,me1=getMesh3D(1)
f=lambda x,y,z: 3*x**2-y**3+z**2+x*y
u=f(q3[:,0],q3[:,1],q3[:,2])
w=[lambda x,y,z: y*np.cos(-(x**2+y**2)/10), lambda x,y,z:
    -x*np.cos(-(x**2+y**2)/10), lambda x,y,z: z/5]
W=np.array([w[0](q3[:,0],q3[:,1],q3[:,2]),
            w[1](q3[:,0],q3[:,1],q3[:,2]),
            w[2](q3[:,0],q3[:,1],q3[:,2])])
mlab.close(all=True)
mlab.figure(1)
mlab4sim.streamline(q3,me3,u,W)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
mlab.colorbar()
mlab.figure(2)
s_options={'visible':True}
sw_options={'normal':(0,0,1),'resolution':6}
st_options={'integration_direction':'both'}
mlab4sim.streamline(q3,me3,u,W,seedtype='plane',linetype='tube',
                    seed_options=s_options,
                    seed_widget_options=sw_options,
                    streamtracer_options=st_options)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
mlab.colorbar()

mlab.figure(3)
sw_options={'center':(0.9,0,1), 'radius':0.1,'phi_resolution':8,
            'theta_resolution':12,'enabled':False}
st_options={'integration_direction':'both'}
mlab4sim.streamline(q3,me3,u,W,seed_widget_options=sw_options,
                    streamtracer_options=st_options,colormap='jet')
sw_options['center']=(0,0,1)
sw_options['radius']=0.3
mlab4sim.streamline(q3,me3,u,W,seed_widget_options=sw_options,
                    streamtracer_options=st_options,colormap='jet')
mlab.scalarbar()
mlab.view(46.6,58,6.7)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)

mlab.figure(4)
sw_options={'origin':(0,-1,0),'point1':(0,-1,2),'point2':(0,1,0),
            'enabled':True,'resolution':6}
st_options={'integration_direction':'both'}
mlab4sim.streamline(q3,me3,u,W,seedtype='plane',
                    seed_widget_options=sw_options,
                    streamtracer_options=st_options,colormap='jet')
mlab.scalarbar()
mlab.view(46.6,58,6.7)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
```

Listing 16: 3D mesh : streamline function