



FC_MAYAVI4MESH package, User's Guide *

François Cuvelier[†]

April 30, 2017

Abstract

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

Contents

1	Introduction	2
1.1	Prerequisite	2
1.2	Installation	2
1.3	Remarks	2
2	function <code>PLOTMESH</code>	3
2.1	2D example	4
2.2	3D example	5
2.3	3D surface example	6
3	function <code>PLOT</code>	6
3.1	2D example	7
3.2	3D example	8
3.3	3D surface example	9
4	function <code>PLOTISO</code>	9
4.1	2D example	10
4.2	3D example	11
4.3	3D surface example	12

*Compiled with Python 3.6.0, packages FC_TOOLS-0.0.9, FC_MAYAVI4MESH-0.0.3, 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.

5	function <code>QUIVER</code>	12
5.1	2D example	13
5.2	3D example	14
5.3	3D surface example	14
6	function <code>SLICEMESH</code>	15
7	function <code>SLICE</code>	16
8	function <code>SLICEISO</code>	17
9	function <code>STREAMLINE</code>	18

1 Introduction

The **experimental** package use Mayavi ($\geq 4.5.0$) 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 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 $0 \leq d \leq dim$ 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 tuple 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,nz])` 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).

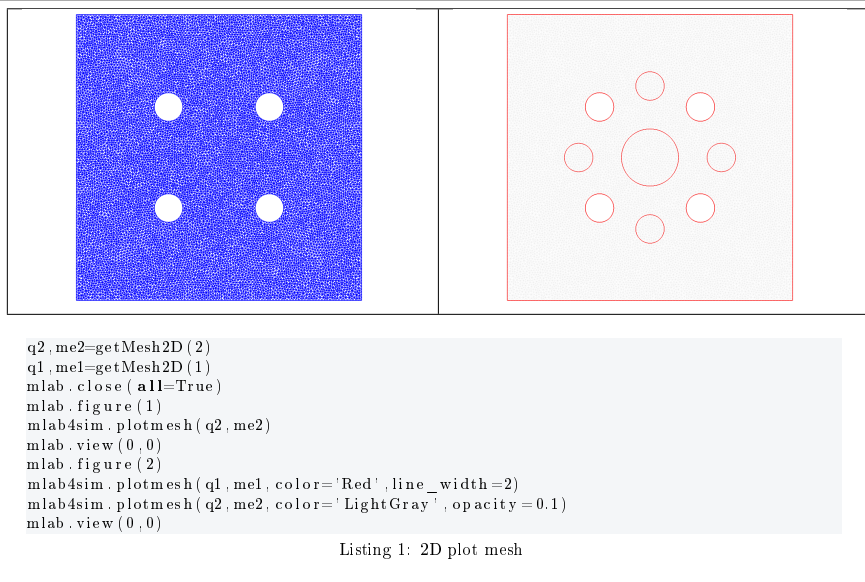
The options of second level depend on the type of elementaries mesh elements to represent.

One can use any option of the following functions according to the type of d -simplex to be represented.

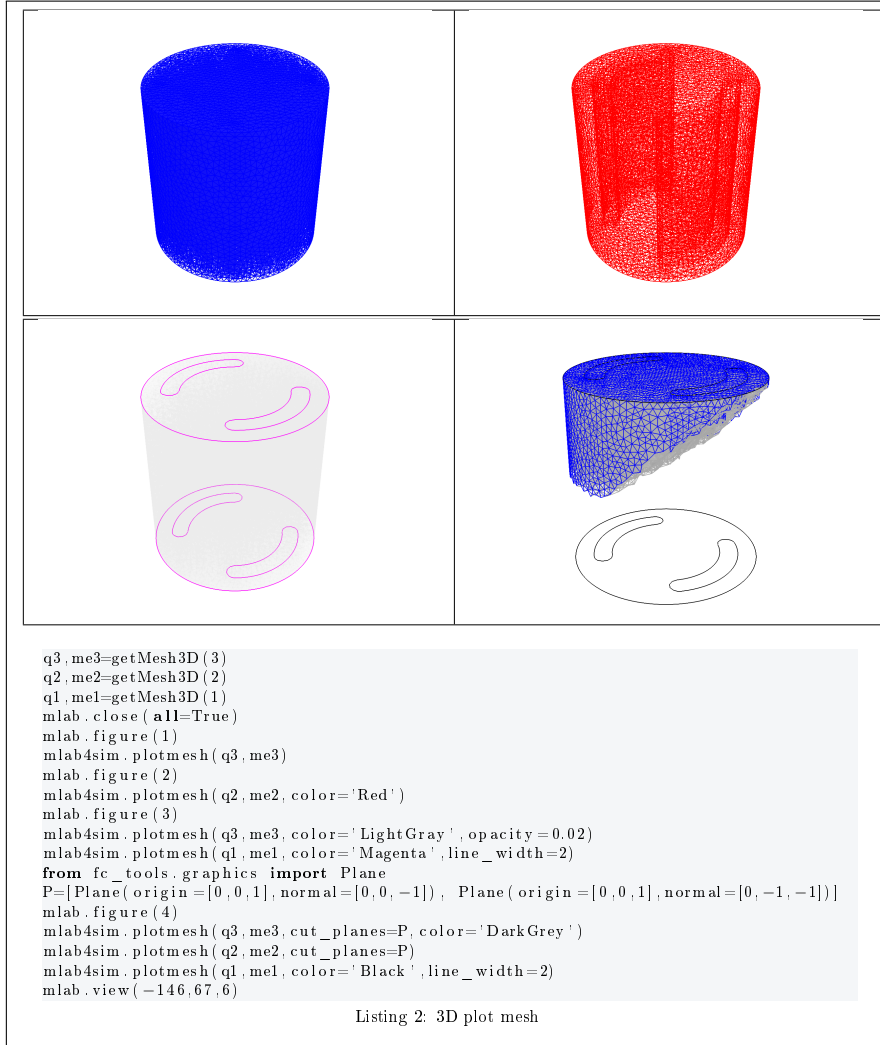
- In dimension 3,
 - if $d == 3$, `mlab.pipeline.surface` function is used with `tvtk.UnstructuredGrid` and `tvtk.Tetra().cell_type`

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

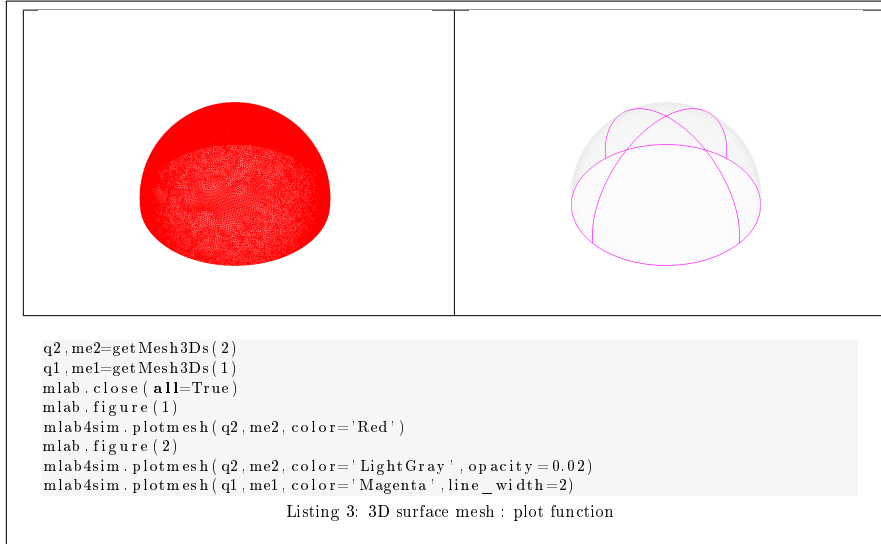
2.1 2D example



2.2 3D example



2.3 3D surface example



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)

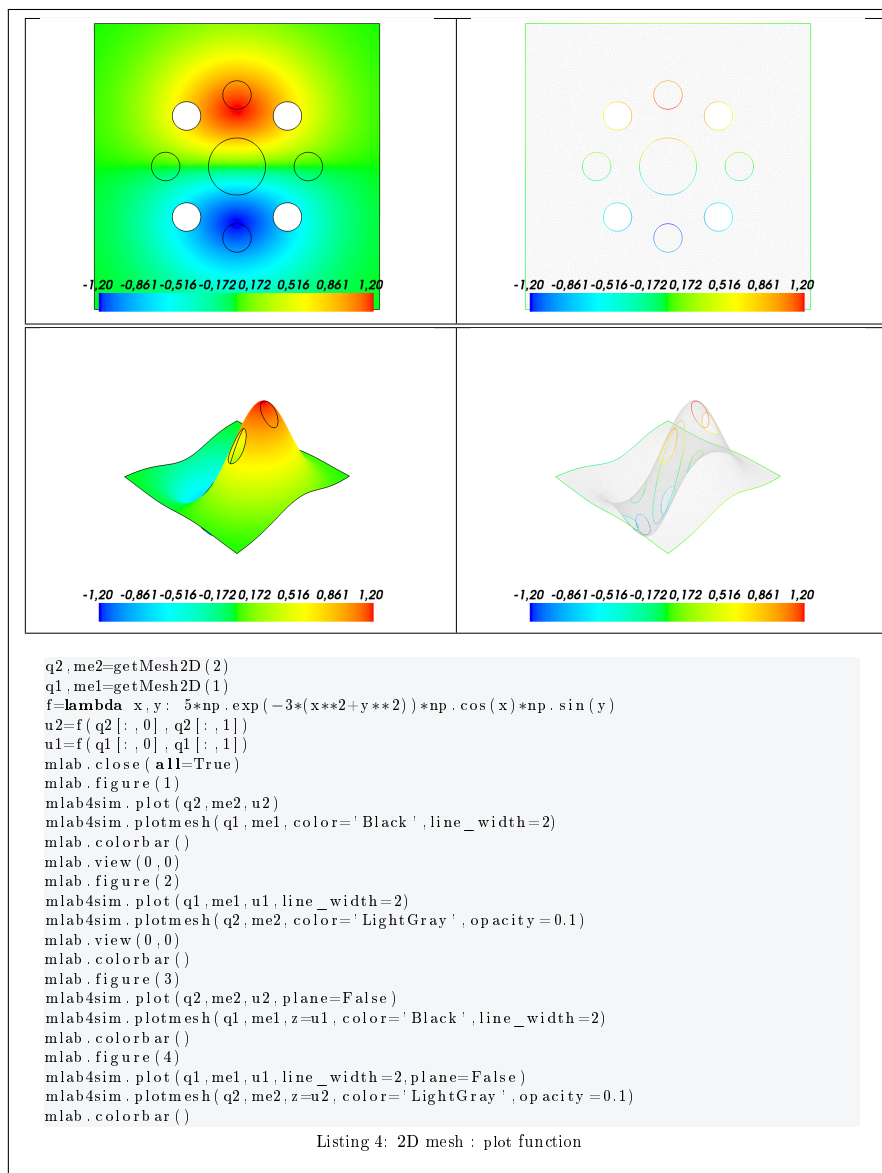
The second level options depend on the type of elementaries mesh elements on which we want to represent datas.

One can use any option of the following functions according to the type of d -simplex.

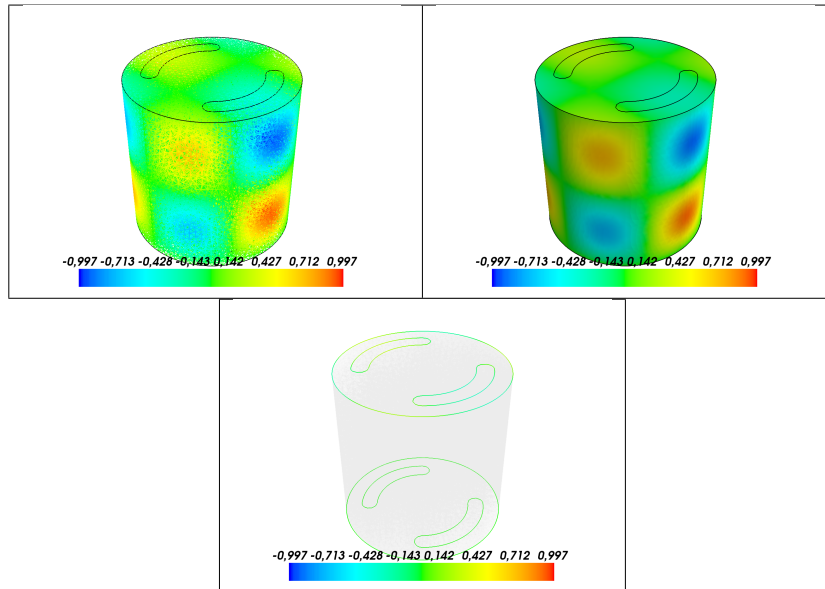
- In dimension 3,
 - if $d == 3$, `mlab.pipeline.surface` function is used with `vtk.UnstructuredGrid` and `vtk.Tetra().cell_type`

- 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



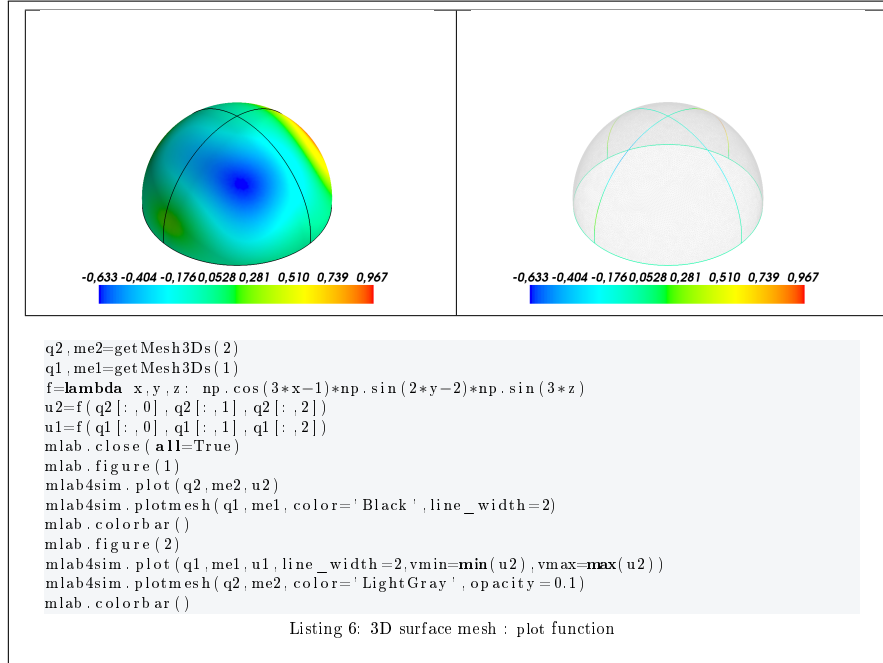
3.2 3D example



```
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='Light Gray', opacity=0.02)
mlab.colorbar()
```

Listing 5: 3D mesh : plot function

3.3 3D surface example



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

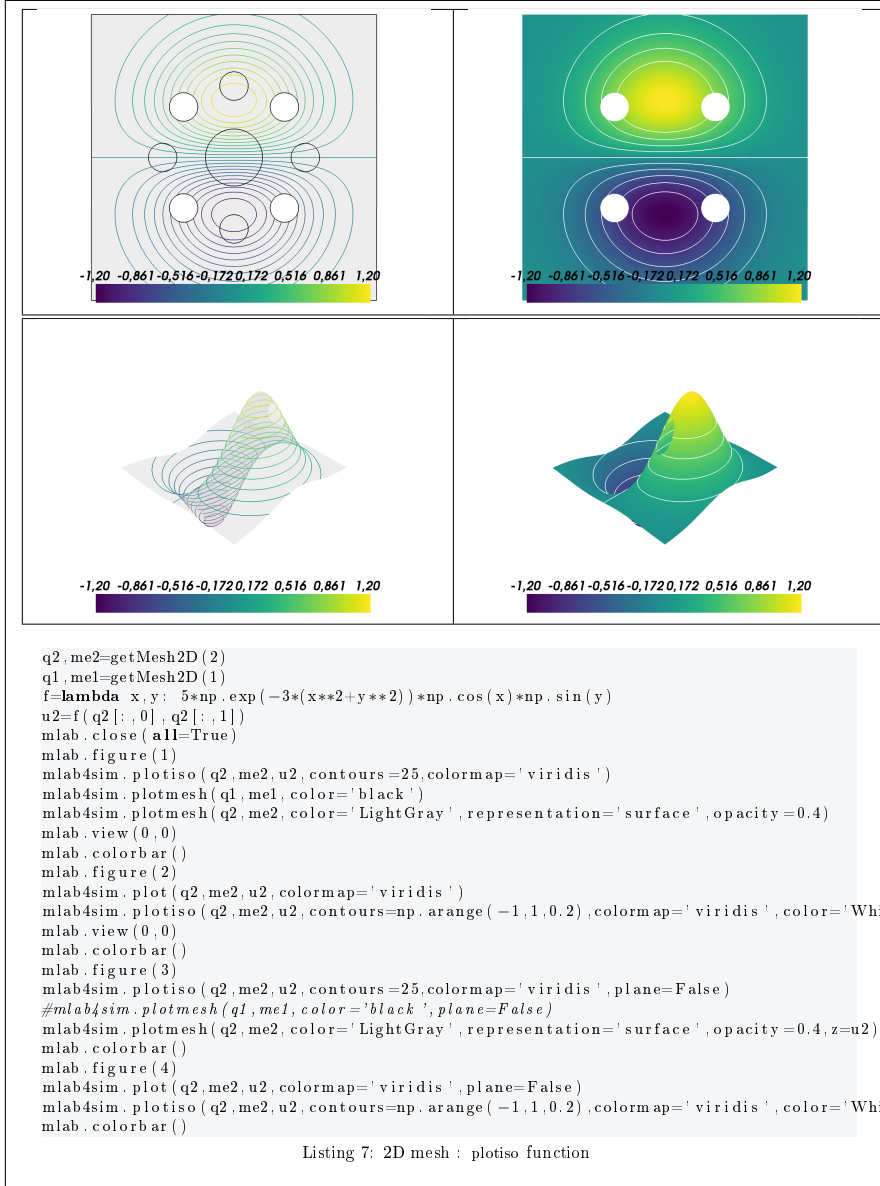
`mlab4sim.plotiso(q,me,u)` displays data `u` on all the 2-dimensional simplices elements as colored isovalues. The data `u` is an 1D-array of size `nq`.

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

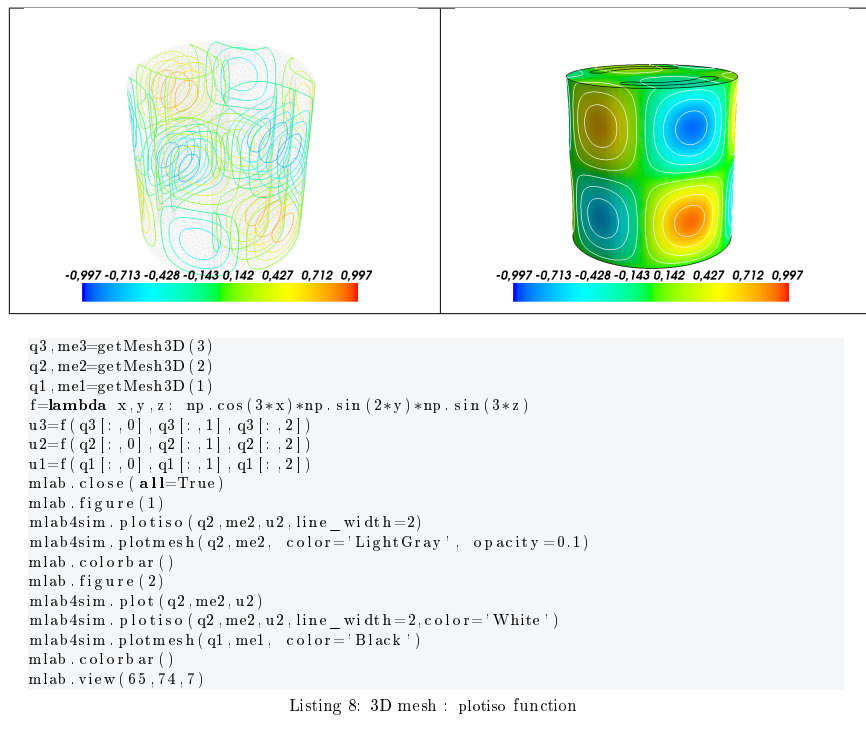
- **contours** : to specify the number of isolines (default : 10) or a list/numpy array of isovalues (default : empty)
- **plane** : if False draw 3D isovalues with data `u` as `z` values. (default : True)
- **color** : to specify one color for all isolines (default : None)

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

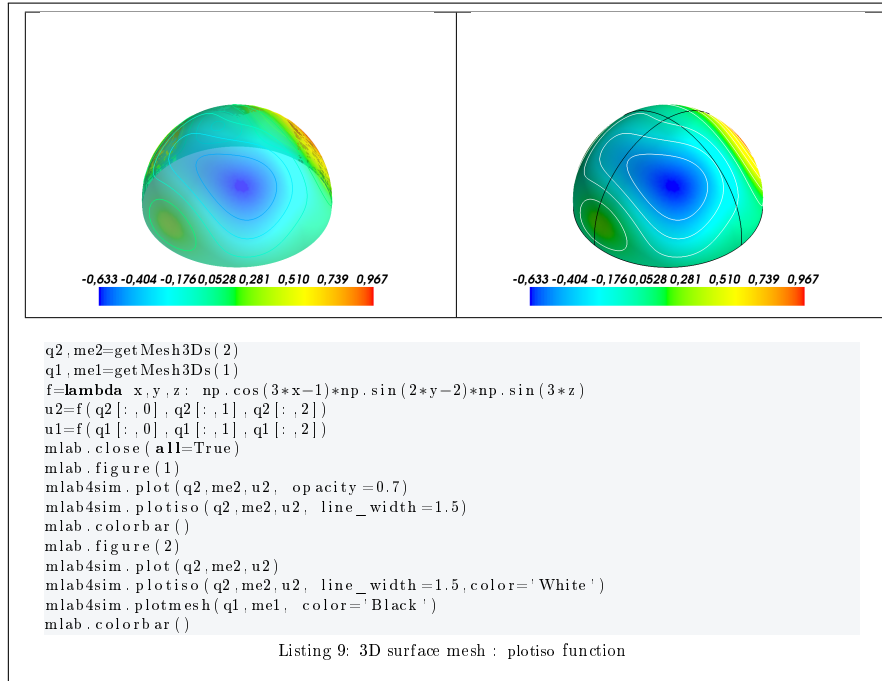
4.1 2D example



4.2 3D example



4.3 3D surface example



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

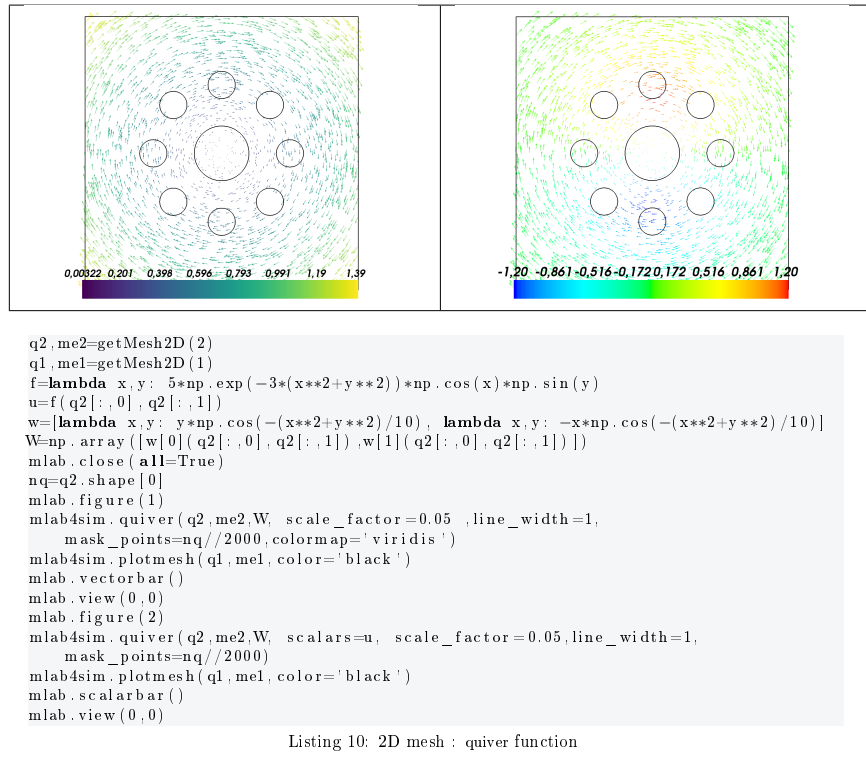
`mlab4sim.quiver(q,me,V)` displays vector field V on each vertices of the d -dimensional simplices elements in dimension $d = 2$ or $d = 3$. The data V is an 2D-array numpy array of size dim -by- nq .

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

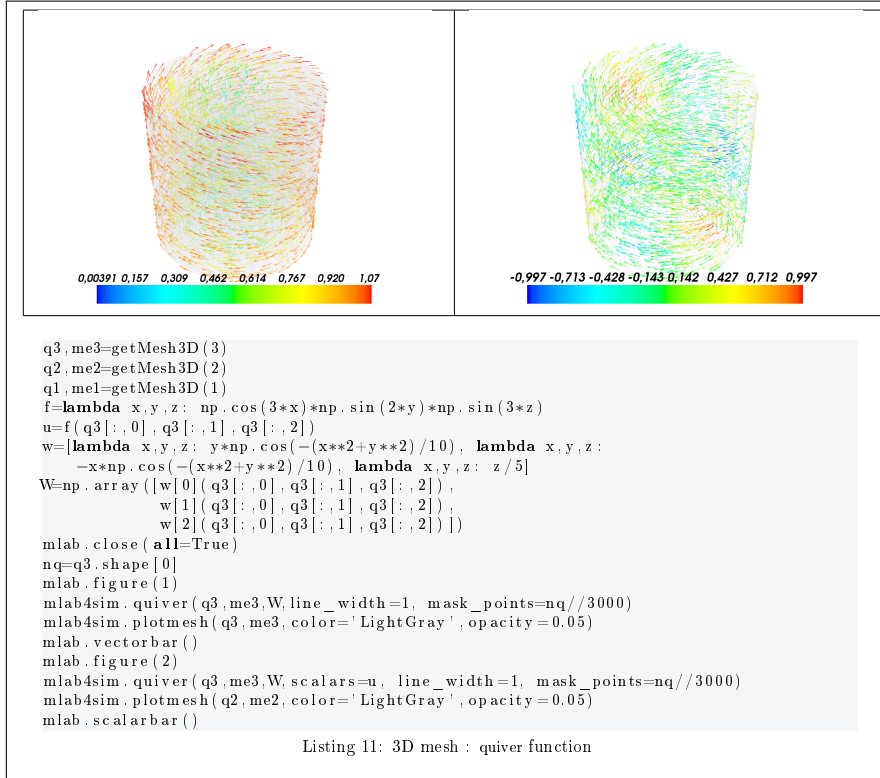
- **scalars** : to set quivers color to a numpy array of size nq (default : empty and use colors of the mesh elements).
- **color** : to specify one color for all quivers.

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

5.1 2D example

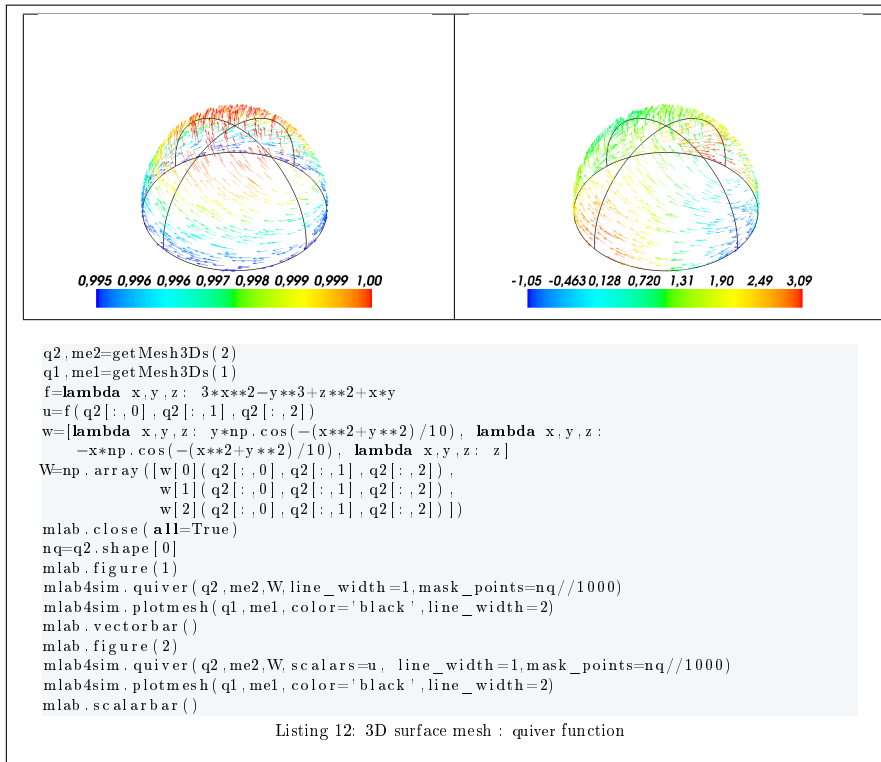


5.2 3D example



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.



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,...)

```

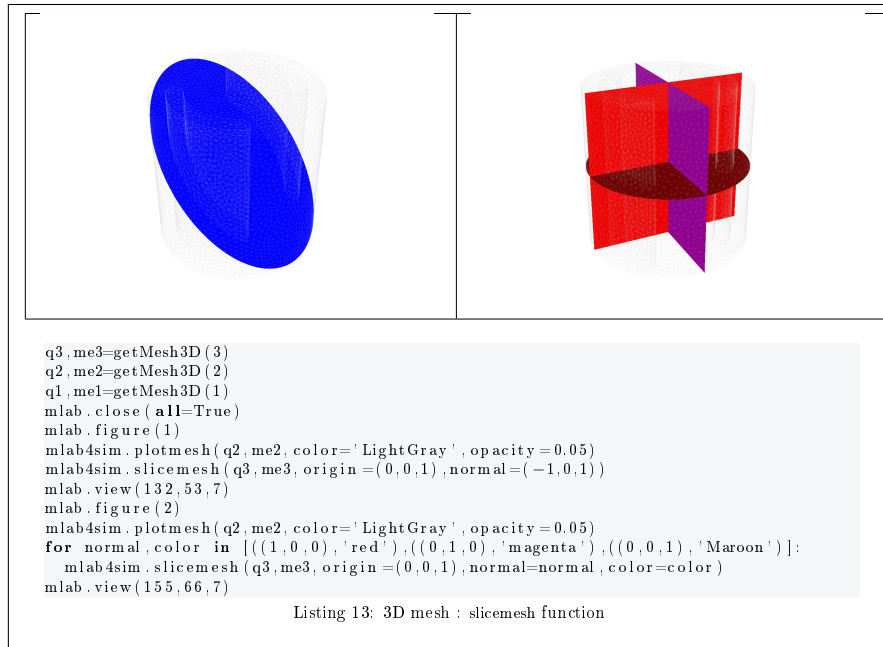
Description

`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`.



7

function `Slice`

The `Slice` function displays data 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,...)

```

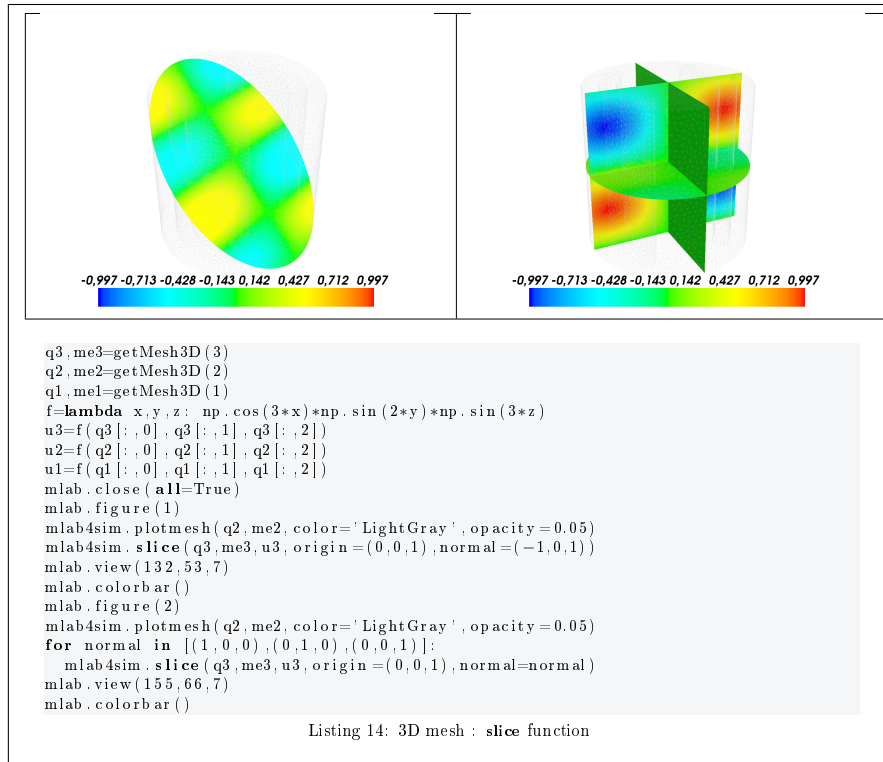
Description

`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`.



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, ...)

```

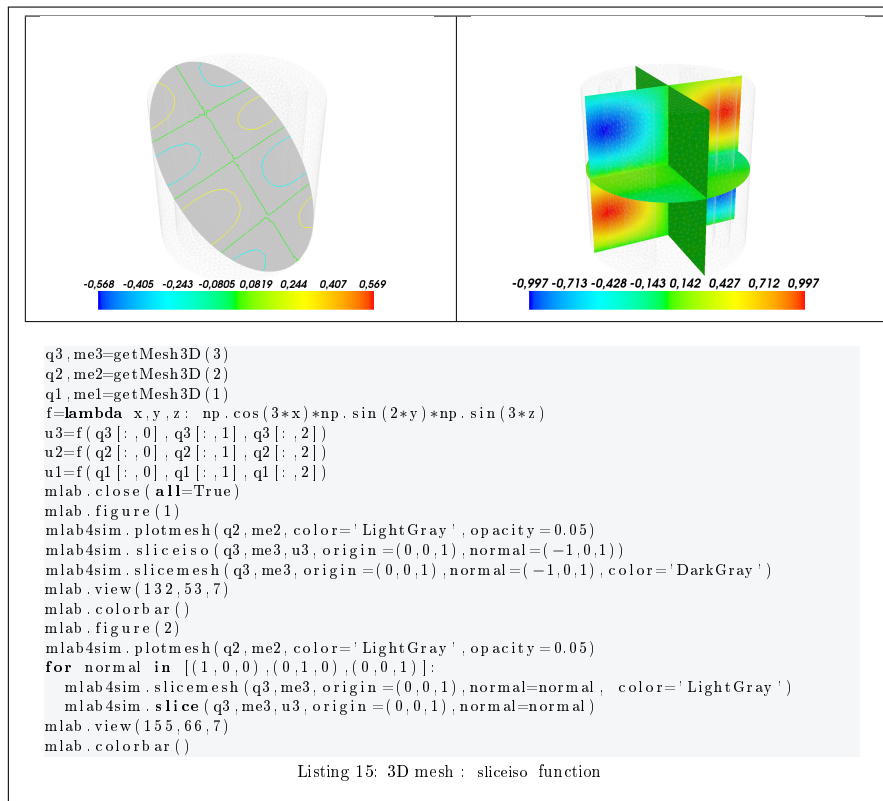
Description

`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`.



9 function **STREAMLINE**

The **STREAMLINE** function allows to draw streamlines for given vector data and colored 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 colored 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.

