mayavi4mesh Python package, User's Guide*

version 0.1.2

F. Cuvelier†

May 13, 2019

Abstract

The mayavi4mesh Python package allows to display simplicial meshes or datas on simplicial meshes by using Mayavi (>= 4.5.0) python package. The simplicial meshes must be given by two arrays : the vertices array and the connectivity array.

0 Contents

*LaTeX manual, revision 0.1.2, compiled with Python 3.7.3, and toolboxes `fc-mayavi4mesh`[0.1.2], `fc-tools`[0.0.22], `fc-meshtools`[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 **experimental** toolbox Python package use Mayavi (>= 4.6.0) 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 supposed to be given by the two arrays `q` and `me`:

- `q` is a *dim*-by-n_q Numpy array, called **vertices array** or **nodes array**, where *dim* is the space dimension (2 or 3) and n_q the number of vertices.
- `me` a $(d+1)$-by-n_{me} Numpy array called, **connectivity array**, where n_{me} is the number of mesh elements and $0 \leqslant d \leqslant dim$ is the simplicial dimension.

So the first vertex of the k-th mesh element is given by `q[:,me[0,k]`, the second vertex is `q[:,me[1,k]`, and so on

For demonstration purpose, we used some simplicial meshes provided by the `fc_meshtools` Python package: they can be load by using the function `getMesh2D`, `getMesh3D` or `getMesh3Ds` of the `fc_meshtools.simplicial` module.

Here is the list of available functions, in `fc_mayavi4mesh.simplicial` module, to represent meshes and to display some information

- `plotmesh` displays a mesh given by a vertices and connectivity arrays,
- `slicemesh` displays intersection of a plane and a 3D mesh given by vertices and connectivity arrays
- `plot_node_indices` displays the indices of the nodes contained in a vertices/nodes array
- `plot_element_indices` displays indices of the simplicial elements contained in a connectivity array,
- `plot_mesh_indices` displays indices of the nodes contained in a vertices/nodes array `q` and indices of a connectivity array `me`.

Theses function are described in Section 2

Here is the list of available functions, in `fc_mayavi4mesh.simplicial` module, for displaying data on meshes:

- `plot` displays datas on a mesh given by vertices and connectivity arrays,
- `plotiso` displays isolines from datas on a mesh given by a vertices and connectivity arrays (only 2-simplicial mesh),
- `quiver` displays vector field on a mesh given by a vertices and connectivity arrays,
- `slice` displays datas on the intersection of a plane and a 3D mesh given by vertices and connectivity arrays,
- `sliceiso` displays isolines of data on the intersection of a plane and a 3D mesh given by vertices and connectivity arrays,
- `streamline` allows to draw streamlines for given vector data and colorized by scalar data on a 3D mesh given by vertices and connectivity arrays.
- `vectors`
- `volume` visualizes scalar fields using volumetric visualization techniques
- `slice_unstructured_grid`

Theses function are described in Section 3

1.1 Installation

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

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

```
$ pip install --user fc_mayavi4mesh
```

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

```
$ sudo pip install fc_mayavi4mesh
```

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

One can note that the **fc_tools** and **fc_meshtools** Python packages are automaticaly installed.

1.2 Remarks

In all this report, for graphical representation, we use `mlab4sim` namespace for graphical functions using Mayavi [1]. There is the python code previously used before any of the listings given thereafter:

```
import numpy as  np
from mayavi import mlab
from fc_meshtools.simplicial import getMesh2D,getMesh3D,getMesh3Ds
 import fc_meshtools.simplicial as mshsim
import fc_mayavi4mesh.simplicial as mlab4sim
from fc_tools.colors import check_color
fig_opts={'size':(1200,1200)}
```

The functions `getMesh2D`, `getMesh3D` and `getMesh3Ds` return a mesh vertices array `q`, a mesh elements connectivity array `me` and an indices array `toGlobal` 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,toGlobal=getMesh3D(2)` returns a 2-simplicial mesh in space dimension $dim = 3$.

2 Mesh functions

2.1 function `plotmesh`

The PLOTMESH function displays a mesh given by a vertices array `q` and a connectivity array `me`.

Syntaxe

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

Description

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

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

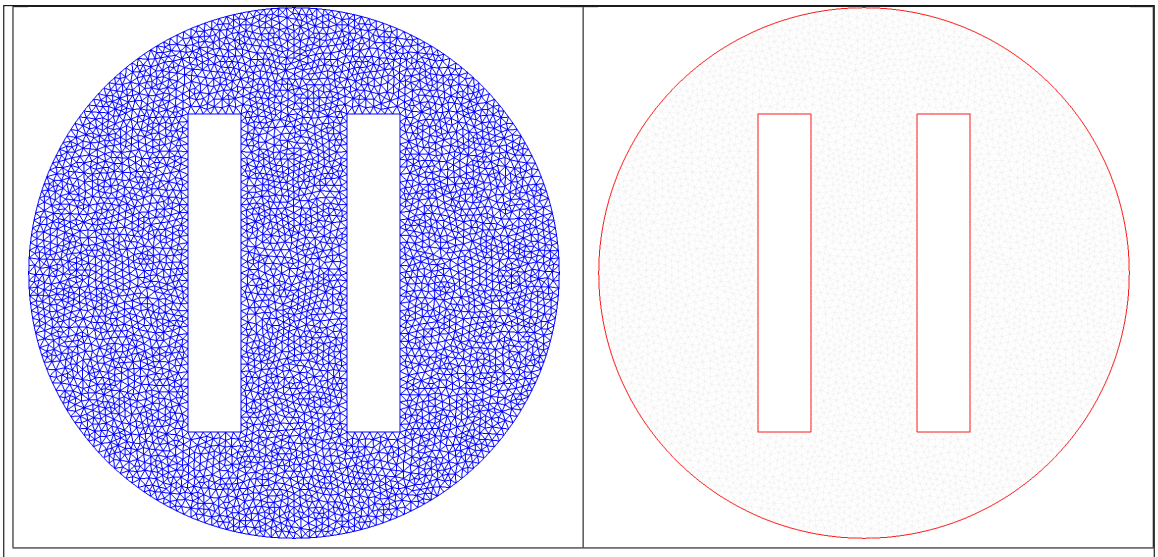
- `color` : specify the mesh color as a RGB tupple or a string color (see check_color ...)

- `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. default : `[]` (no cut).

In dimension 2 with a 2-simplicial(triangle) mesh, the second level options are those of the `mlab.triangular_mesh` function. Otherwise, they are those of the `mlab.pipeline.surface` function.

The `obj` output is the graphical object created by the function.

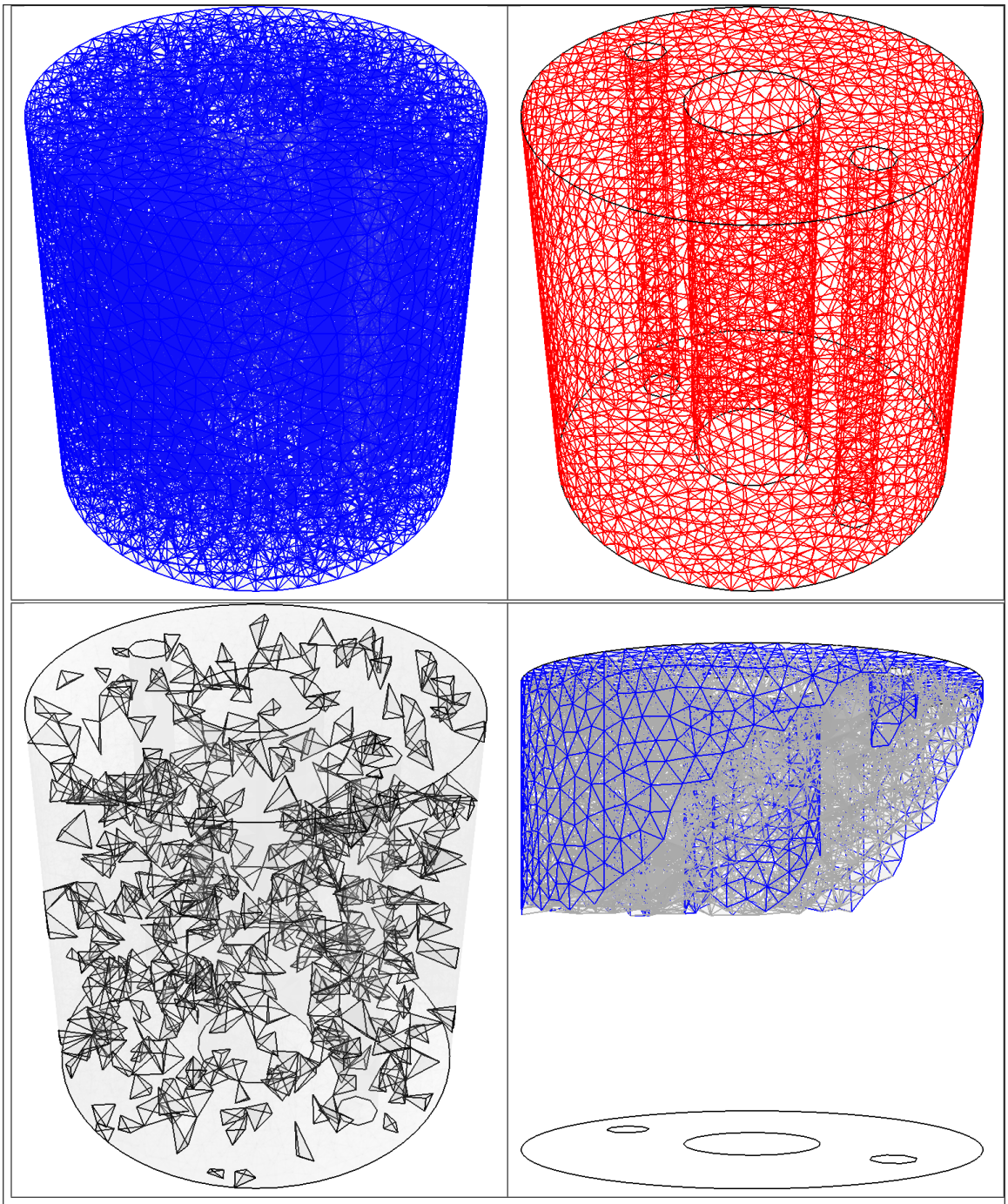
2.1.1 2D example



```
q2,me2=getMesh2D(2)[:2]
q1,me1=getMesh2D(1)[:2]
mlab.close(all=True)
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q2,me2)
mlab.view(0,0)
mlab.figure(2,**fig_opts)
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

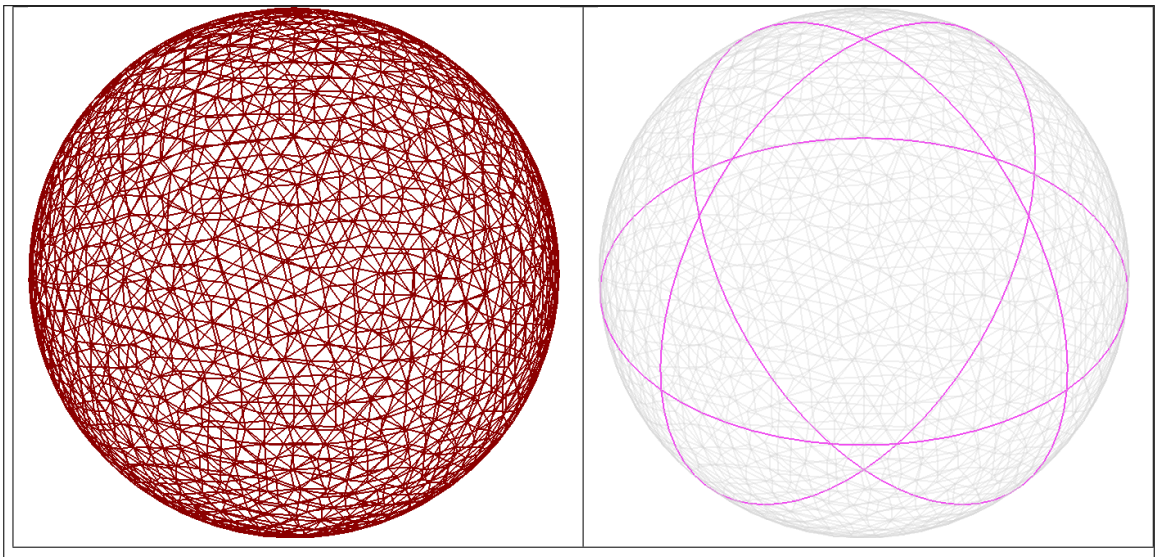
2.1.2 3D example



```
q3,me3=getMesh3D(3)[:2]
q2,me2=getMesh3D(2)[:2]
q1,me1=getMesh3D(1)[:2]
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q3,me3)
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='Red')
mlab4sim.plotmesh(q1,me1,color='k',line_width=2)
mlab.figure(3,**fig_opts)
indices=np.arange(0,me3.shape[1],50)
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.02)
mlab4sim.plotmesh(q3,me3,color='k',indices=indices)
mlab4sim.plotmesh(q1,me1,color='k',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,**fig_opts)
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(-146,90,6)
```

Listing 2: 3D plot mesh

2.1.3 3D surface example



```
q2,me2=getMesh3Ds(2)[:2]
q1,me1=getMesh3Ds(1)[:2]
mlab.close(all=True)
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='darkred')
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.2)
mlab4sim.plotmesh(q1,me1,color='Magenta',line_width=2)
```

Listing 3: 3D surface mesh : plot function

2.2 function `slicemesh`

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

Syntaxe

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

Description

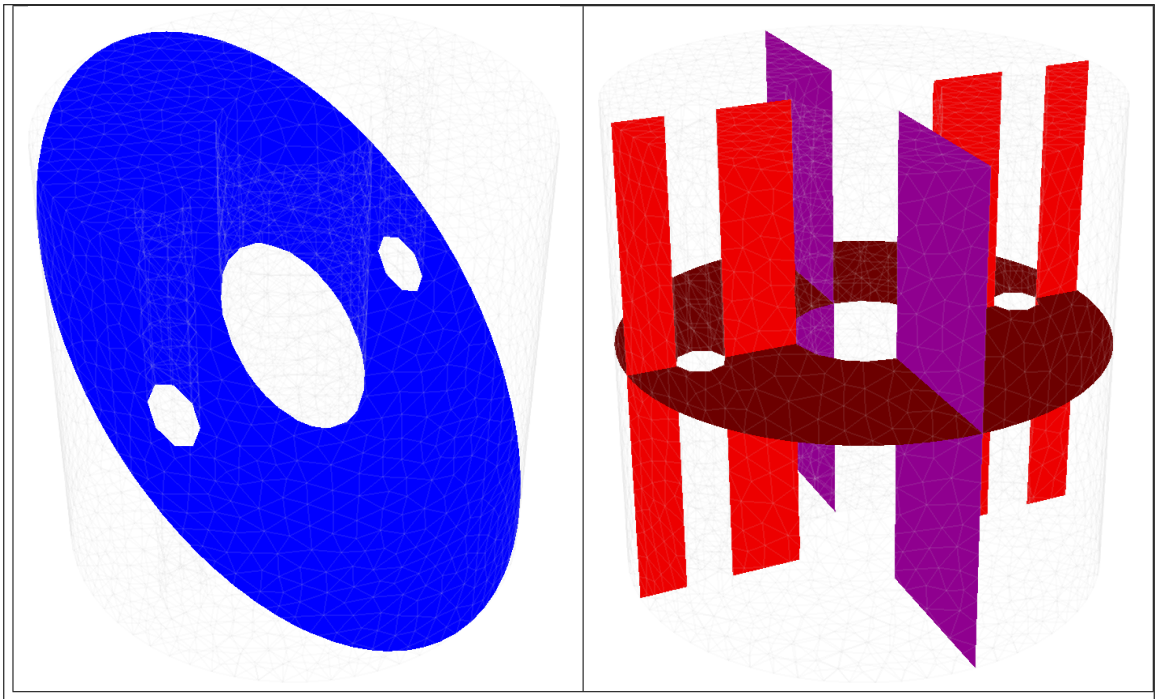
`mlab4sim.slicemesh(q,me,)` displays intersection of a plane and all the 3-dimensional simplices elements. By default the plane is given by an *origin* point $(0,0,0)$ which lies on it and a *normal* vector $(0,0,1)$ which is orthogonal to the plane.

`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`.

The `obj` output is the graphical object created by the function.



```
q3,me3=getMesh3D(3)[:2]
q2,me2=getMesh3D(2)[:2]
q1,me1=getMesh3D(1)[:2]
mlab.figure(1,**fig_opts)
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,**fig_opts)
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 4: 3D mesh : `slicemesh` function

2.3 function `plot_node_indices`

The `plot_node_indices` function displays the indices of the nodes contained in a vertices/nodes array by using `mlab.pipeline.labels` function.

Syntaxe

```
obj=mlab4sim.plot_node_indices(q)
obj=mlab4sim.plot_node_indices(q,Key=Value, ...)
```

Description

`mlab4sim.plot_node_indices(q)` displays all the indices of the nodes contained in the `q` vertices/nodes array. `q` is an array of shape `(2,nq)` or `(3,nq)`. For each node `q[:,k]`, the `k` index is written in `q[:,k]` position on the current figure, extended to $z = 0$ if `q.shape[0]==2`.

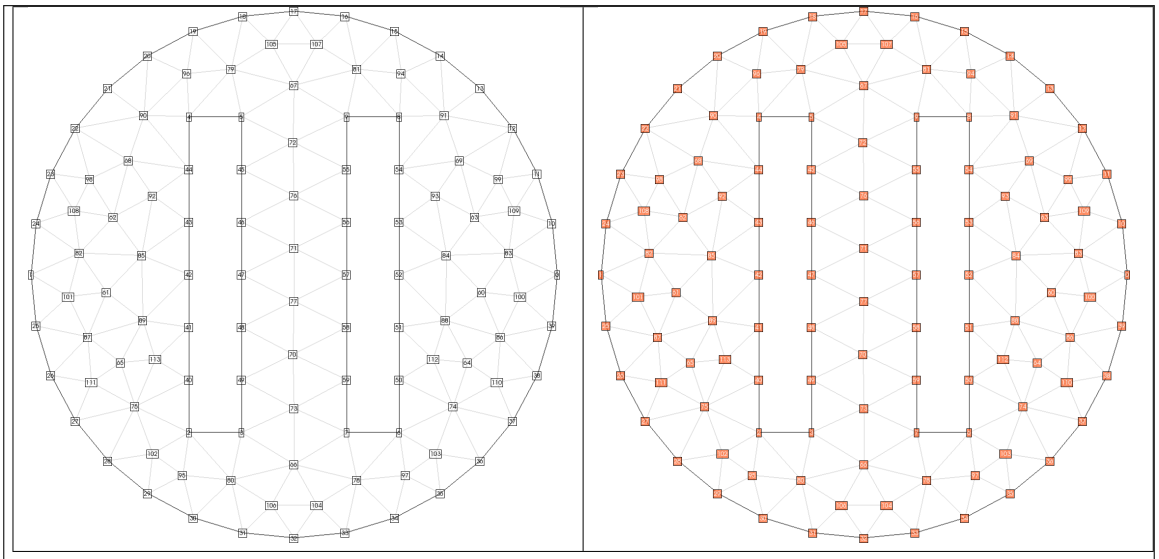
`mlab4sim.plot_node_indices(q,Key=Value, ...)` specifies function options using one or more `Key`,`Value` pair arguments. Options `Key` are

- `indices` : to specify a part of the indices, so `Value` is a numpy 1D-array with unique elements in $[\![0, \texttt{nq}[\![$.
- `toGlobal` :
- `property` : to specify `property` of the `mayavi.modules.labels.Labels` object created by the function. `Value` is a dictionary where keys could be (explanation from `text_property.py` in `tvtk_classes.zip`)
  - `color` The text color as a RGB tuple (default is `'Black' (0,0,0)`).

- `bold`: Enable/disable text bolding. boolean, (default `False`),
- `italic`: Enable/disable text italic. boolean (default `False`),
- `shadow`: Enable/disable text shadow. boolean (default `False`),
- `shadow_offset`: default `array([ 1, -1])`,
- `justification` Set the horizontal justification to `'left'`, `'centered'`(default), or `'right'`.
- `vertical_justification`: Set the vertical justification to `'bottom'`, `'centered'`(default), or `'top'`.
- `background_color`: Set the background color as a RGB tuple. Default is `'White' (1,1,1)`,
- `background_opacity`: The background opacity. 1.0 is totally opaque and 0.0 is completely transparent (default 0.6).
- `font_family`: Set the font family. Supports legacy three font family system. If the symbolic constant `VTK_FONT_FILE` is returned by `get_font_family()`, the string returned by `get_font_file()` must be an absolute filepath to a local freetype compatible font (default `'arial'`).
- `font_file` The absolute filepath to a local file containing a freetype-readable font if `get_font_family()` return `VTK_FONT_FILE`. The result is undefined for other values of `get_font_family()` (default `None`).
- `font_size` Set the font size (in points) (default `12`).
- `use_tight_bounding_box` If this property is on, text is aligned to drawn pixels not to font metrix. If the text does not include descents, the bounding box will not extend below the baseline. This option can be used to get centered labels. It does not work well if the string changes as the string position will move around. (default `False`)
- `frame` Enable/disable text frame. boolean (default `True`),
- `frame_color` The frame color as a tuple (default is `'Black' (0,0,0)`).
- `frame_width` Set the width of the frame. The width is expressed in pixels. The default is `1` pixel.
- `line_offset` Set the (extra) spacing between lines, expressed as a text height multiplication factor (default 2.0),
- `line_spacing` Set the text's opacity. 1.0 is totally opaque and 0.0 is completely transparent (default 2.0).
- `opacity` Set the text's opacity. 1.0 is totally opaque and 0.0 is completely transparent (default 1.0).
- `orientation` Set the text's orientation (in degrees) (default 0.0)

The `obj` output is the graphical object created by the function.

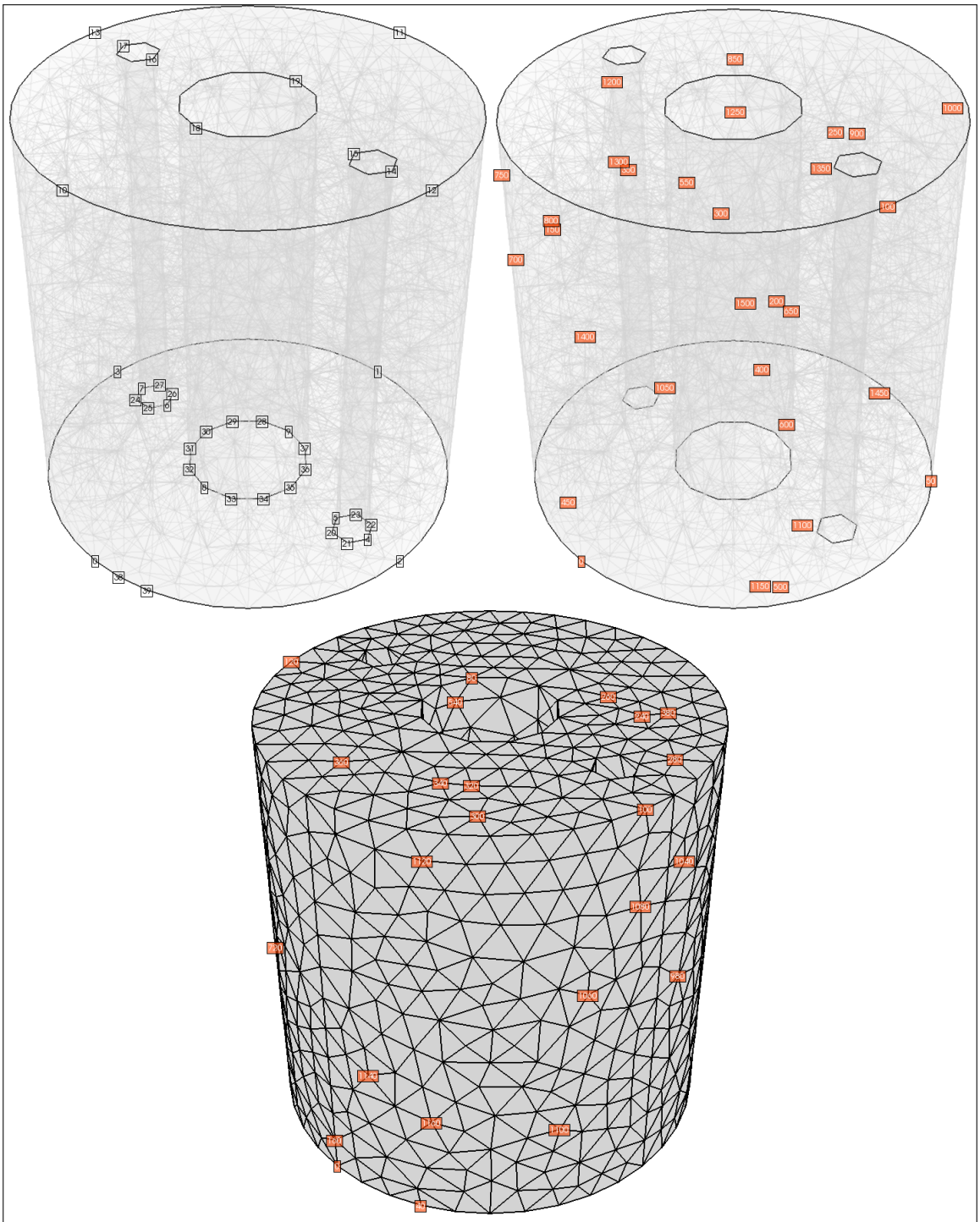
2.3.1 2D example



```
q2,me2=getMesh2D(2,small=True)[:2]
q1,me1=getMesh2D(1,small=True)[:2]
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.4)
mlab4sim.plotmesh(q1,me1,color='k',line_width=2,opacity=0.5)
mlab4sim.plot_node_indices(q2)
mlab.view(0,0)
node_prop={'color': check_color('White'),
           'frame': True,
           'frame_color': check_color('Black'),
           'background_color': check_color('OrangeRed'),
           'background_opacity': 0.6}
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.4)
mlab4sim.plotmesh(q1,me1,color='k',line_width=2,opacity=0.5)
mlab4sim.plot_node_indices(q2, property=node_prop)
mlab.view(0,0)
```

Listing 5: `plot_node_indices` function on 2D example

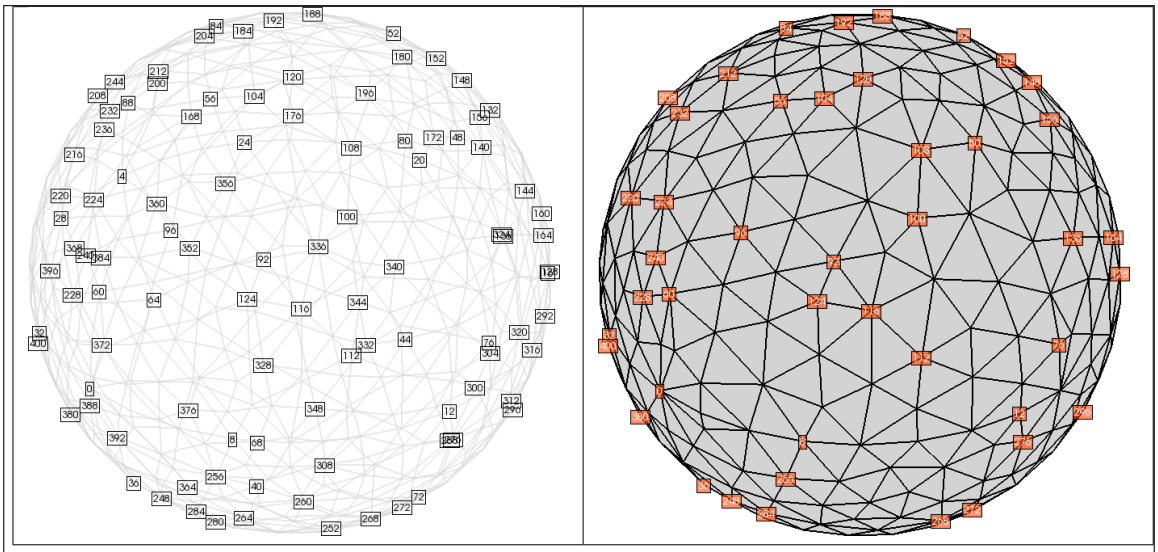
2.3.2 3D example



```
q3,me3,toG3=getMesh3D(3,small=True)
q2,me2,toG2=getMesh3D(2,small=True)
q1,me1=getMesh3D(1,small=True)[:2]
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.3)
mlab4sim.plotmesh(q1,me1,color='k')
# Only plot the 40 first node indices
mlab4sim.plot_node_indices(q3,indices=np.arange(40))
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.3)
mlab4sim.plotmesh(q1,me1,color='k')
node_prop= {'color':check_color('White'),
          'frame':True,
          'frame_color':check_color('Black'),
          'background_color':check_color('OrangeRed'),
          'background_opacity':0.6}
mlab4sim.plot_node_indices(q3,indices=np.arange(0,q3.shape[1],50),property=node_prop)
mlab.figure(3,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=1,representation='surface',edge_visibility=True)
mlab4sim.plotmesh(q1,me1,color='k')
mlab4sim.plot_node_indices(q2,indices=np.arange(0,q2.shape[1],20),property=node_prop)
```

Listing 6: `plot_node_indices` function on 3D example

2.3.3 3D surface example



```
q2,me2=getMesh3Ds(2,small=True)[:2]
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.3)
mlab4sim.plot_node_indices(q2,indices=np.arange(0,q2.shape[1],4))
node_prop= {'color':check_color('White'),
          'frame':True,
          'frame_color':check_color('Black'),
          'background_color':check_color('OrangeRed'),
          'background_opacity':0.6}
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=1,representation='surface',edge_visibility=True)
mlab4sim.plot_node_indices(q2,property=node_prop,indices=np.arange(0,q2.shape[1],4))
```

Listing 7: `plot_node_indices` function on 3D surface example

2.4 function `plot_element_indices`

The `plot_element_indices` function displays indices of the simplicial elements contained in a connectivity array by using `mlab.pipeline.labels` function.

Syntaxe

```
obj=mlab4sim.plot_element_indices(q,me)
obj=mlab4sim.plot_element_indices(q,me,Key=Value, ...)
```

Description

`mlab4sim.plot_element_indices(q,me)` displays all the indices of the `d`-simplices contained in the `me` connectivity array. `q` is an array of shape `(2,nq)` or `(3,nq)` and `me` is an array of shape `(d+1,nme)`. For each element `me[:,k]`, the `k` index is written in the barycenter of the element (extended to $z = 0$ if `q.shape[0]==2`).

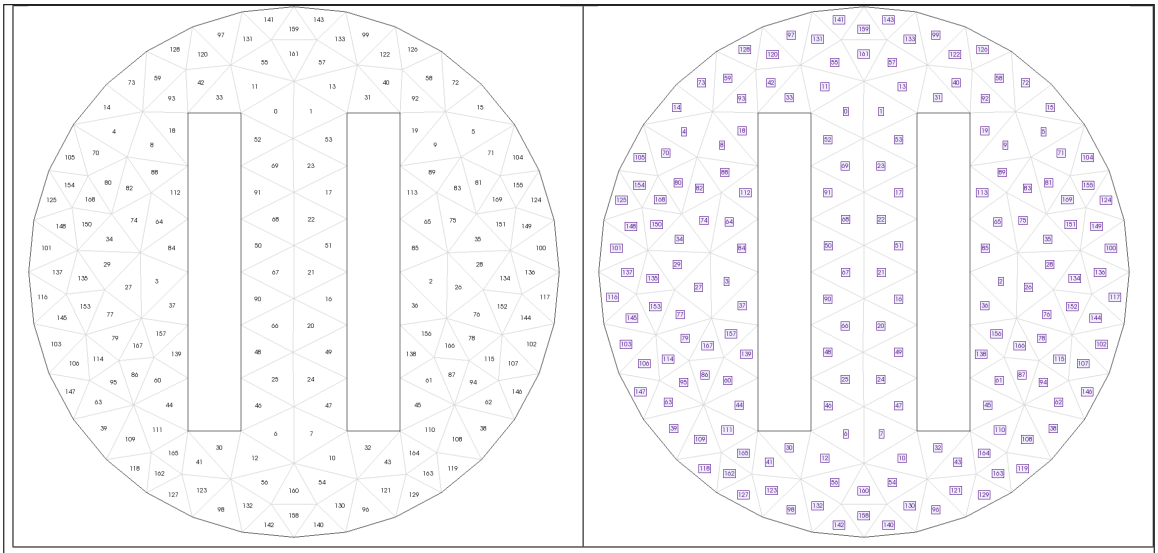
`mlab4sim.plot_element_indices(q,meKey=Value, ...)` specifies function options using one or more `Key`,`Value` pair arguments. Options `Key` are

- `indices` : to specify a part of the indices, so `Value` is a numpy 1D-array with unique elements in $[\![0, \texttt{nme}[\![$.
- `visible_enabled`: if `True`, all selected mesh elements are displayed with color given by `mesh_color` value
- `mesh_color`: set the color of the mesh elements as a RGB tuple or (default is black `(0,0,0)`.
- `toGlobal` :
- `property` : to specify `property` of the `mayavi.modules.labels.Labels` object created by the function. `Value` is a dictionary where keys could be (explanation from `tvtk_classes.zip/text_property.py`)

- bold: Enable/disable text bolding. boolean, (default False),
- italic: Enable/disable text italic. boolean (default False),
- shadow: Enable/disable text shadow. boolean (default False),
- shadow_offset: default array([1, -1]),
- font_family: Set the font family. Supports legacy three font family system. If the symbolic constant VTK_FONT_FILE is returned by get_font_family(), the string returned by get_font_file() must be an absolute filepath to a local freetype compatible font (default 'arial').
- justification Set the horizontal justification to 'left', 'centered'(default), or 'right'.
- vertical_justification: Set the vertical justification to 'bottom', 'centered'(default), or 'top'.
- background_color: Set the background color as a RGB tuple. Default is 'GhostWhite' (0.972, 0.972, 1.0),
- background_opacity: The background opacity. 1.0 is totally opaque and 0.0 is completely transparent (default 0.6).
- color The text color as a RGB tuple (default is 'Black' (0,0,0)),
- font_file The absolute filepath to a local file containing a freetype-readable font if get_font_family() return VTK_FONT_FILE. The result is undefined for other values of get_font_family() (default None).
- font_size Set the font size (in points) (default 12).
- use_tight_bounding_box If this property is on, text is aligned to drawn pixels not to font metrix. If the text does not include descents, the bounding box will not extend below the baseline. This option can be used to get centered labels. It does not work well if the string changes as the string position will move around. (default False)
- frame Enable/disable text frame. boolean (default False),
- frame_color The frame color as a tuple (default is 'Black' (0,0,0)).
- frame_width Set the width of the frame. The width is expressed in pixels. The default is 1 pixel.
- line_offset Set the (extra) spacing between lines, expressed as a text height multiplication factor (default 2.0),
- line_spacing Set the text's opacity. 1.0 is totally opaque and 0.0 is completely transparent (default 2.0).
- opacity Set the text's opacity. 1.0 is totally opaque and 0.0 is completely transparent (default 1.0).
- orientation Set the text's orientation (in degrees) (default 0.0)

The obj output is the graphical object created by the function.

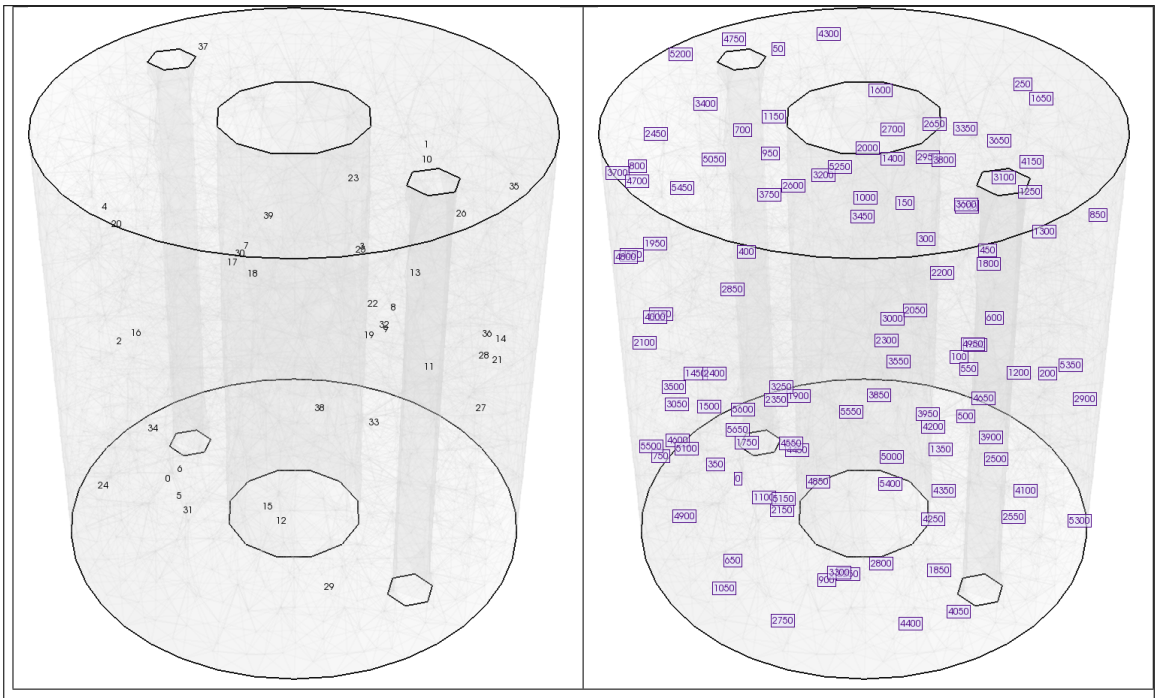
2.4.1 2D example



```
q2,me2=getMesh2D(2,small=True)[:2]
q1,me1=getMesh2D(1,small=True)[:2]
mlab.close(all=True)
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.4)
mlab4sim.plotmesh(q1,me1,color='k',line_width=2,opacity=0.5)
mlab4sim.plot_element_indices(q2,me2)
mlab.view(0,0)
elt_prop={'color':check_color('Indigo'),
          'frame':True,
          'frame_color':check_color('Indigo'),
          'background_color':check_color('Lavender'),
          'background_opacity':0.6}
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.4)
mlab4sim.plotmesh(q1,me1,color='k',line_width=2,opacity=0.5)
mlab4sim.plot_element_indices(q2,me2,property=elt_prop)
mlab.view(0,0)
```

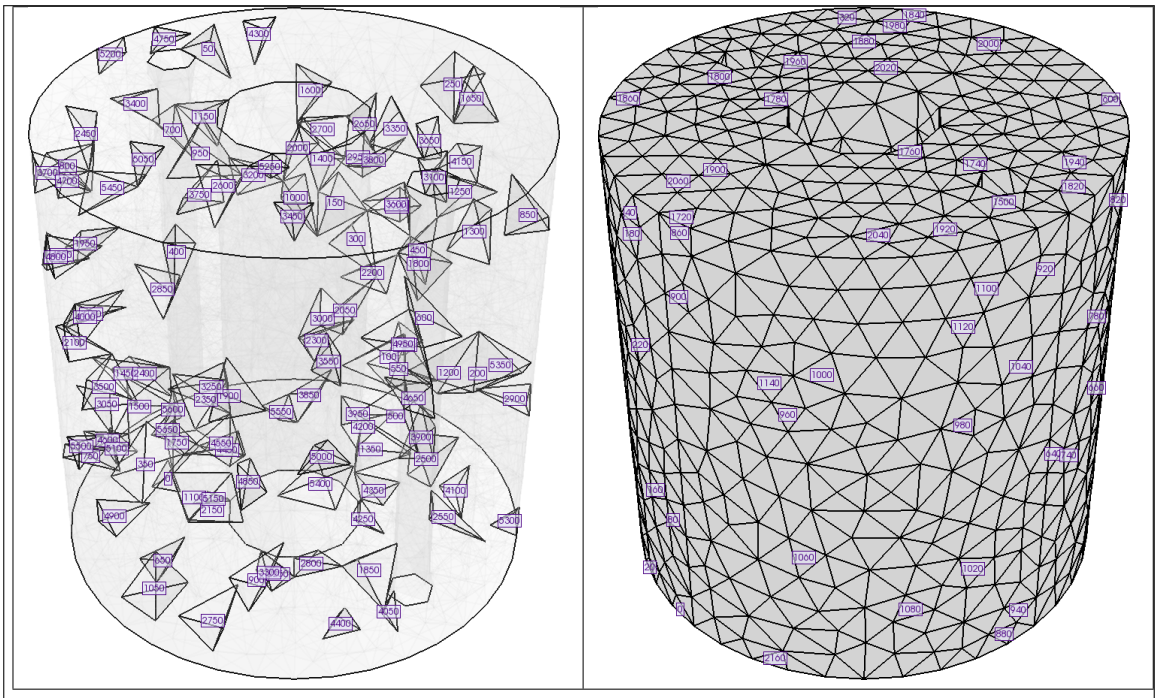
Listing 8: `plot_element_indices` function on 2D example

2.4.2 3D examples



```
q3,me3,toG3=getMesh3D(3,small=True)
q2,me2,toG2=getMesh3D(2,small=True)
q1,me1=getMesh3D(1,small=True)[:2]
mlab.close(all=True)
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.1)
mlab4sim.plotmesh(q1,me1,color='k')
# Only plot the 40 first element indices
mlab4sim.plot_element_indices(q3,me3,indices=np.arange(40))
elt_prop={'color':check_color('Indigo'),
          'frame':True,
          'frame_color':check_color('Indigo'),
          'background_color':check_color('Lavender'),
          'background_opacity':0.6}
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.1)
mlab4sim.plotmesh(q1,me1,color='k')
mlab4sim.plot_element_indices(q3,me3,indices=np.arange(0,me3.shape[1],50), property=elt_prop)
```

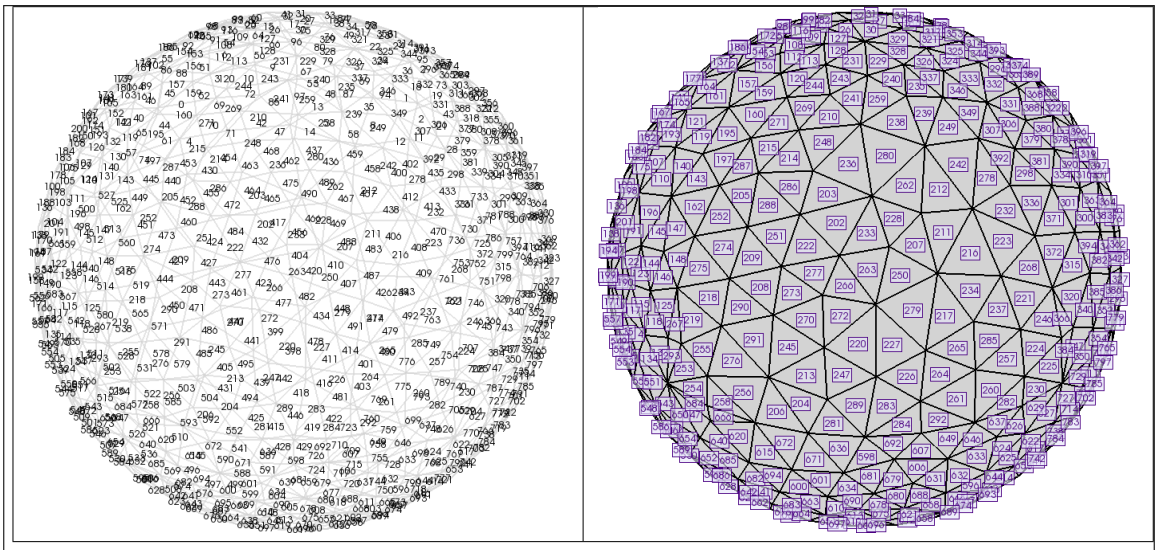
Listing 9: `plot_element_indices` function: first 3D example



```
q3,me3,toG3=getMesh3D(3,small=True)
q2,me2,toG2=getMesh3D(2,small=True)
q1,me1=getMesh3D(1,small=True)[:2]
elt_prop={'color':check_color('Indigo'),
          'frame':True,
          'frame_color':check_color('Indigo'),
          'background_color':check_color('Lavender'),
          'background_opacity':0.6}
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.1)
mlab4sim.plotmesh(q1,me1,color='k')
mlab4sim.plot_element_indices(q3,me3,indices=np.arange(0,me3.shape[1],50),
   property=elt_prop,visible_enabled=True)
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=1,representation='surface',
   edge_visibility=True)
mlab4sim.plotmesh(q1,me1,color='k')
mlab4sim.plot_element_indices(q2,me2,indices=np.arange(0,me2.shape[1],20),property=elt_prop)
```

Listing 10: `plot_element_indices` function: second example

2.4.3 3D surface example



```
q2,me2=getMesh3Ds(2,small=True)[:2]
mlab.close(all=True)
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.4)
mlab4sim.plot_element_indices(q2,me2)
elt_prop={'color':check_color('Indigo'),
          'frame':True,
          'frame_color':check_color('Indigo'),
          'background_color':check_color('Lavender'),
          'background_opacity':0.6}
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=1, representation='surface',
   edge_visibility=True)
mlab4sim.plot_element_indices(q2,me2,property=elt_prop)
```

Listing 11: `plot_element_indices` function on 3D surface example

2.5 function `plot_mesh_indices`

The `plot_mesh_indices` function displays indices of the nodes contained in a vertices/nodes array `q` and indices of a connectivity array `me` by using `plot_node_indices` and `plot_element_indices` functions.

Syntaxe

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

Description

`mlab4sim.plot_mesh_indices(q,me)` displays all the indices of the `d`-simplices contained in the `me` connectivity array and all the indices of the `q` nodes/vertices array by using respectively the function `mlab4sim.plot_element_indices(q,me)` and function `mlab4sim.plot_node_indices(q)`.

`mlab4sim.plot_mesh_indices(q,me,Key=Value, ...)` specifies function options using one or more `Key`,`Value` pair arguments. Options `Key` are

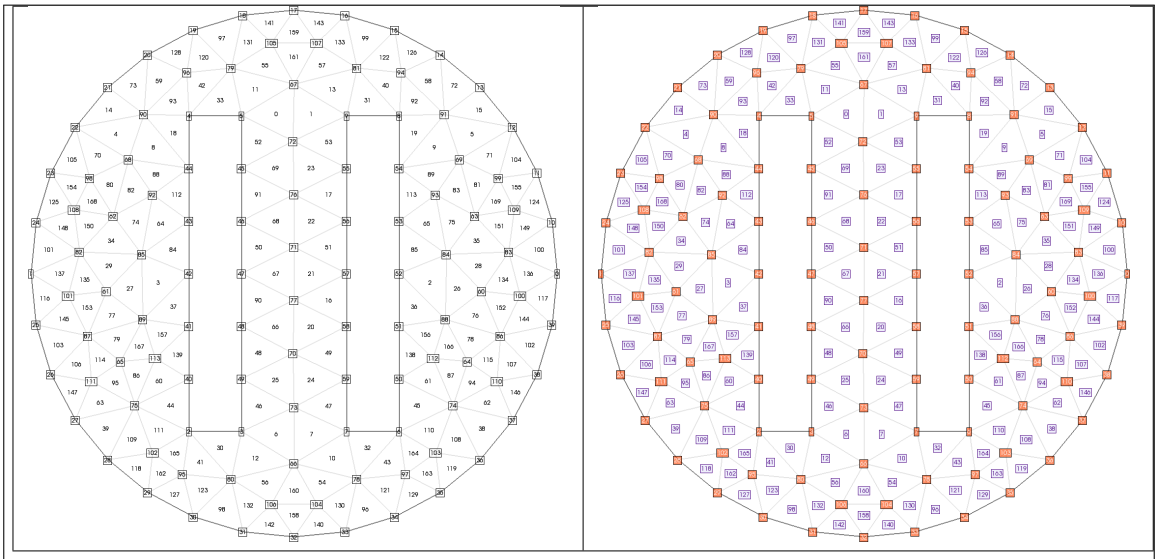
- `element_indices` : used to specify the `indices` option of the `mlab4sim.plot_element_indices` function.
- `element_property` : used to specify the `property` option of the `mlab4sim.plot_element_indices` function.
- `node_indices` : used to specify the `indices` option of the `mlab4sim.plot_node_indices` function.
- `node_property` : used to specify the `property` option of the `mlab4sim.plot_node_indices` function.

- `visible_enabled`: directly passed to `mlab4sim.plot_element_indices` function.
- `mesh_color`: directly passed to `mlab4sim.plot_element_indices` function.
- `toGlobal` : directly passed to `mlab4sim.plot_node_indices` function.

The behavior of the function with respect to the selected options is the following:

- If `element_indices` and `node_indices` are not specified, then all the element indices and all the node indices are plotted.
- If `element_indices` is specified and `node_indices` is not specified, then the elements with given indices are plotted as well as node indices which are vertices of the selected elments.
- If `element_indices` is not specified and `node_indices` is specified, then the nodes with given indices are plotted as well as elements indices which have at least one of the selected nodes as vertices.
- If `element_indices` and `node_indices` are specified, then the given element indices and all the given node indices are plotted.

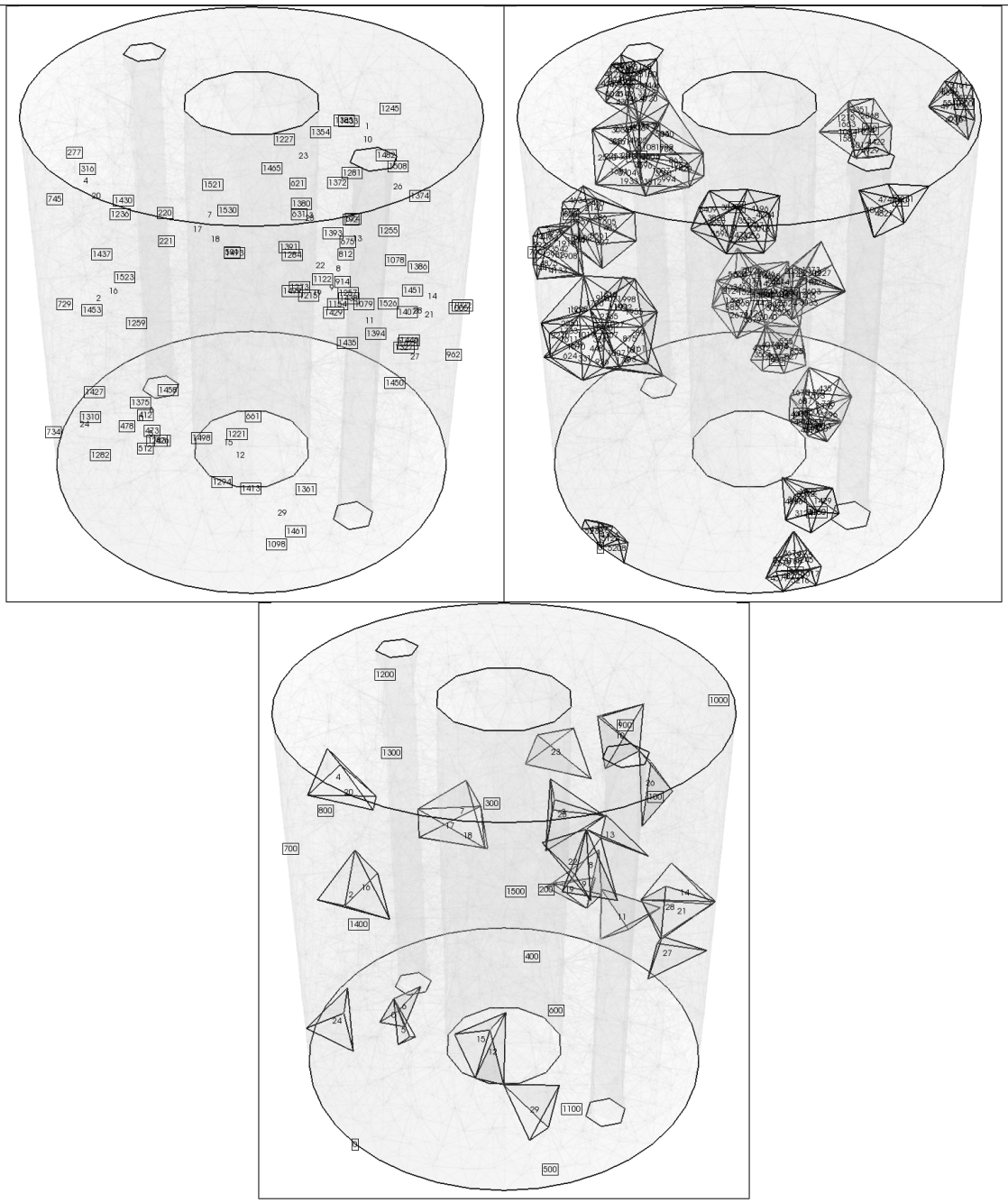
2.5.1 2D example



```
q2,me2=getMesh2D(2,small=True)[:2]
q1,me1=getMesh2D(1,small=True)[:2]
mlab.close(all=True)
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.4)
mlab4sim.plot_mesh_indices(q2,me2)
mlab4sim.plotmesh(q1,me1,color='k',line_width=2,opacity=0.5)
mlab4sim.plot_mesh_indices(q2,me2)
mlab.view(0,0)
elt_prop={'color':check_color('Indigo'),
        'frame':True,
        'frame_color':check_color('Indigo'),
        'background_color':check_color('Lavender'),
        'background_opacity':0.6}
node_prop= {'color':check_color('White'),
          'frame':True,
          'frame_color':check_color('Black'),
          'background_color':check_color('OrangeRed'),
          'background_opacity':0.6}
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.4)
mlab4sim.plotmesh(q1,me1,color='k',line_width=2,opacity=0.5)
mlab4sim.plot_mesh_indices(q2,me2,element_property=elt_prop,node_property=node_prop)
mlab.view(0,0)
```

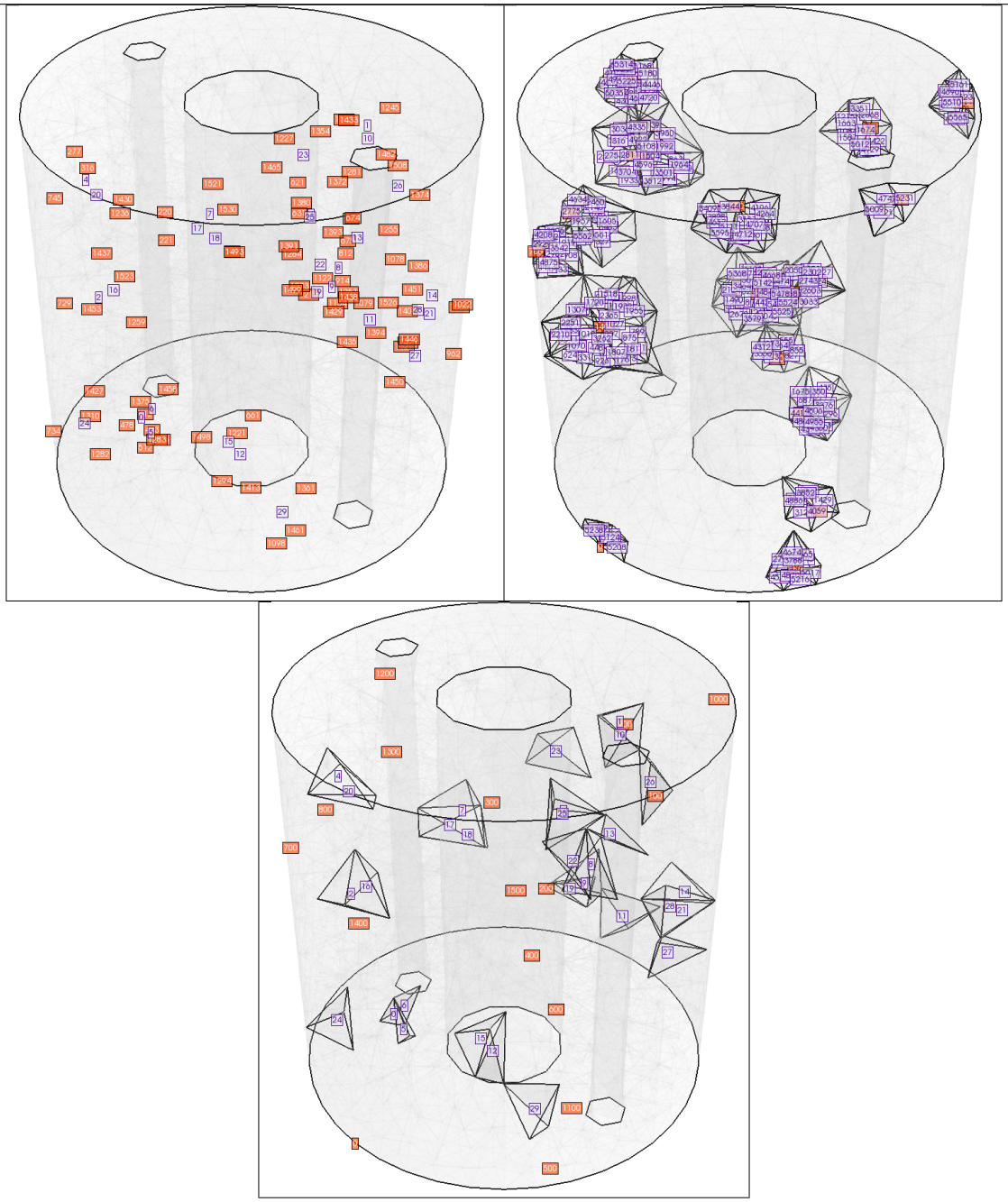
Listing 12: `plot_mesh_indices` function on 2D example

2.5.2 3D example



```
q3,me3,toG3=getMesh3D(3,small=True)
q2,me2,toG2=getMesh3D(2,small=True)
q1,me1=getMesh3D(1,small=True)[:2]
node_idx=np.arange(0,q3.shape[1],100)
elt_idx=np.arange(30)
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.1)
mlab4sim.plotmesh(q1,me1,color='k')
# Only plot the 40 first element indices
mlab4sim.plot_mesh_indices(q3,me3,element_indices=elt_idx)
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.1)
mlab4sim.plotmesh(q1,me1,color='k')
mlab4sim.plot_mesh_indices(q3,me3,node_indices=node_idx,visible_enabled=True)
mlab.figure(3,**fig_opts)
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.1)
mlab4sim.plotmesh(q1,me1,color='k')
mlab4sim.plot_mesh_indices(q3,me3,node_indices=node_idx,element_indices=elt_idx,
    visible_enabled=True)
```

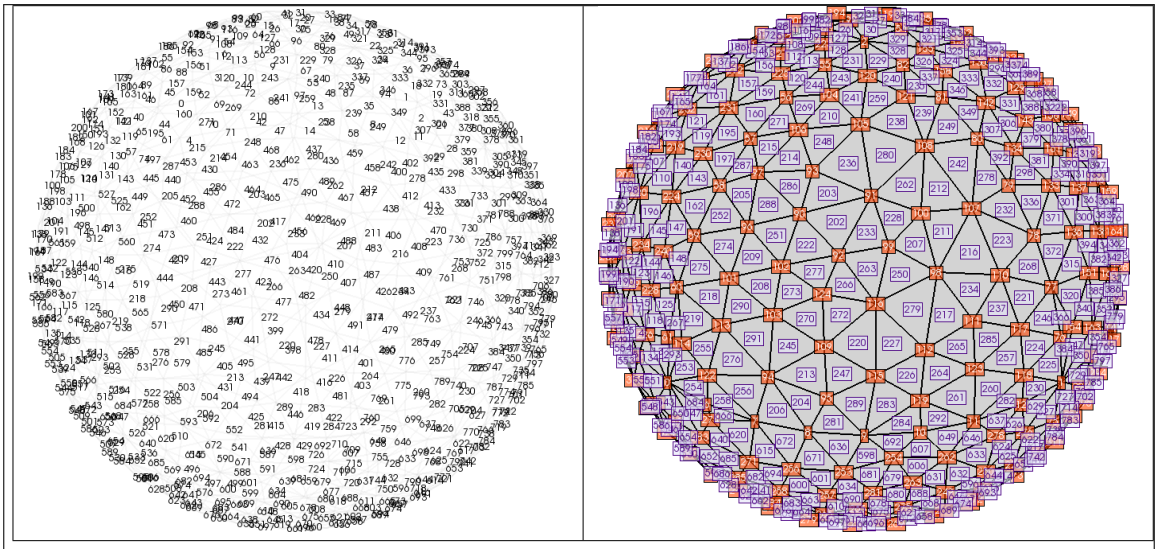
Listing 13: `plot_mesh_indices` function: first 3D example



```
q3,me3,toG3=getMesh3D(3,small=True)
q2,me2,toG2=getMesh3D(2,small=True)
q1,me1=getMesh3D(1,small=True)[:2]
node_idx=np.arange(0,q3.shape[1],100)
elt_idx=np.arange(30)
elt_prop={'color':check_color('Indigo'),'frame':True, 'frame_color':check_color('Indigo'),
    'background_color':check_color('Lavender'), 'background_opacity':0.6}
node_prop= {'color':check_color('White'),'frame':True, 'frame_color':check_color('Black'),
    'background_color':check_color('OrangeRed'), 'background_opacity':0.6}
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.1)
mlab4sim.plotmesh(q1,me1,color='k')
# Only plot the 40 first element indices
mlab4sim.plot_mesh_indices(q3,me3,element_indices=elt_idx,element_property=elt_prop,
    node_property=node_prop)
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.1)
mlab4sim.plotmesh(q1,me1,color='k')
mlab4sim.plot_mesh_indices(q3,me3,node_indices=node_idx,element_property=elt_prop,
    node_property=node_prop,visible_enabled=True)
mlab.figure(3,**fig_opts)
mlab4sim.plotmesh(q3,me3,color='LightGray',opacity=0.1)
mlab4sim.plotmesh(q1,me1,color='k')
mlab4sim.plot_mesh_indices(q3,me3,node_indices=node_idx,element_indices=elt_idx,
    element_property=elt_prop,node_property=node_prop,visible_enabled=True)
```

Listing 14: `plot_mesh_indices` function: second 3D example

2.5.3 3D surface example



```
q2,me2=getMesh3Ds(2,small=True)[:2]
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.1)
mlab4sim.plot_element_indices(q2,me2)
elt_prop={'color':check_color('Indigo'),'frame':True, 'frame_color':check_color('Indigo'),
    'background_color':check_color('Lavender'), 'background_opacity':0.6}
node_prop= {'color':check_color('White'),'frame':True, 'frame_color':check_color('Black'),
    'background_color':check_color('OrangeRed'), 'background_opacity':0.6}
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=1,representation='surface',edge_visibility=True)
mlab4sim.plot_mesh_indices(q2,me2,element_property=elt_prop,node_property=node_prop)
```

Listing 15: `plot_mesh_indices` function on 3D surface example

2.6 function `plot_normal_faces`

The `plot_normal_faces` function displays for each d-simplex element of the mesh the $d+1$ normals to its faces where the mesh is given by a vertices/nodes array `q` and by a connectivity array `me`.

Syntaxe

```
out=mlab4sim.plot_normal_faces(q,me)
out=mlab4sim.plot_normal_faces(q,me,Key=Value, ...)
```

Description

`mlab4sim.plot_normal_faces(q,me)` displays for each d-simplex element of the mesh the $d+1$ normals to its faces `mlab4sim.plot_element_indices(q,me)`. Each of the $d+1$ normals to the faces of an element is plotted in its barycenter with a line connecting to the node/vertex not in the face. A normal and the associated line are the same color.

`mlab4sim.plot_normal_faces(q,me,Key=Value, ...)` specifies function options using one or more `Key`,`Value` pair arguments. Options `Key` are

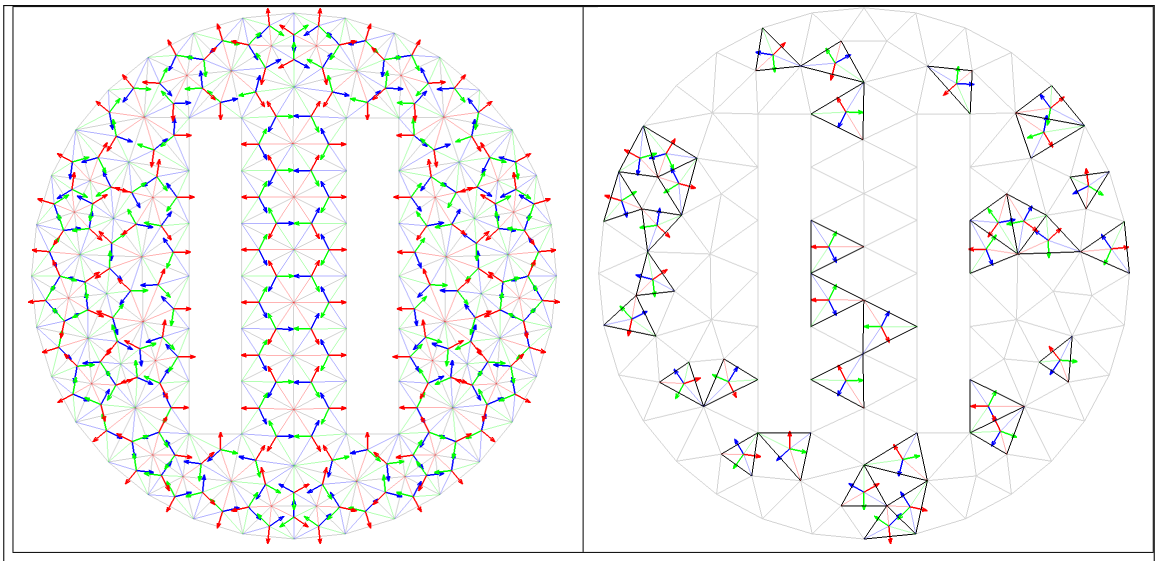
- `barycenters`: to specify barycenters of the mesh elements as an dim-by-n_{me} numpy array. Default is computed by using `barycenters` function of the `fc_meshtools.simplicial` module.
- `normals`: to specify the $d+1$ normals to the faces of mesh elements as an `d+1`-by-dim-by-n_{me} numpy array. Default is computed by using `NormalFaces` function of the `fc_meshtools.simplicial` module.
- `indices` : to specify a part of the indices, so `Value` is a numpy 1D-array with unique elements in $[\![0, nme[\![$.
- `visible_enabled`: if `True`, all selected mesh elements are displayed with color given by `mesh_color` value

- `mesh_color`: set the color of the mesh elements as a RGB tuple or (default is black `(0,0,0)`.
- `colors`: to specify RGB colors of the $d+1$ normals as a $(d+1)$-by-3 numpy array.

The `out` output is a 3-tuple if `visible_enabled` is `True`, otherwise a 2-tuple.

- `out[0]` is a list of length $d+1$ which contains the graphical objects representing the $d+1$ normals on each selected mesh elements.
- `out[1]` is a list of length $d+1$ which contains the graphical objects representing the $d+1$ lines on each selected mesh elements.
- `out[2]` (if `visible_enabled` is `True`) is the graphical objects representing the selected mesh elements.

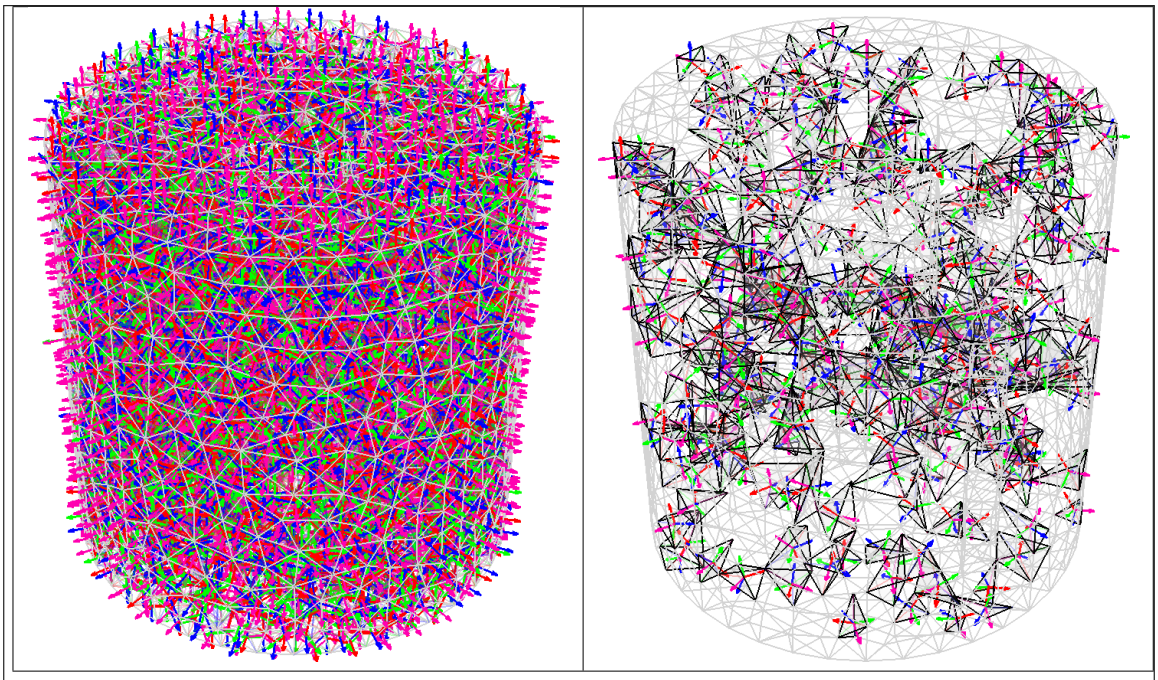
2.6.1 2D example



```
q2,me2=getMesh2D(2,small=True)[:2]
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray')
mlab4sim.plot_normal_faces(q2,me2)
mlab.view(0,0)
Ba=mshsim.barycenters(q2,me2)
G=mshsim.GradBaCo(q2,me2)
Normals=mshsim.NormalFaces(q2,me2,G)
indices=np.arange(0,me2.shape[1],5)
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray')
mlab4sim.plot_normal_faces(q2,me2,indices=indices,visible_enabled=True,barycenters=Ba,normals=Normals)
mlab.view(0,0)
```

Listing 16: `plot_normal_faces` function on 2D example

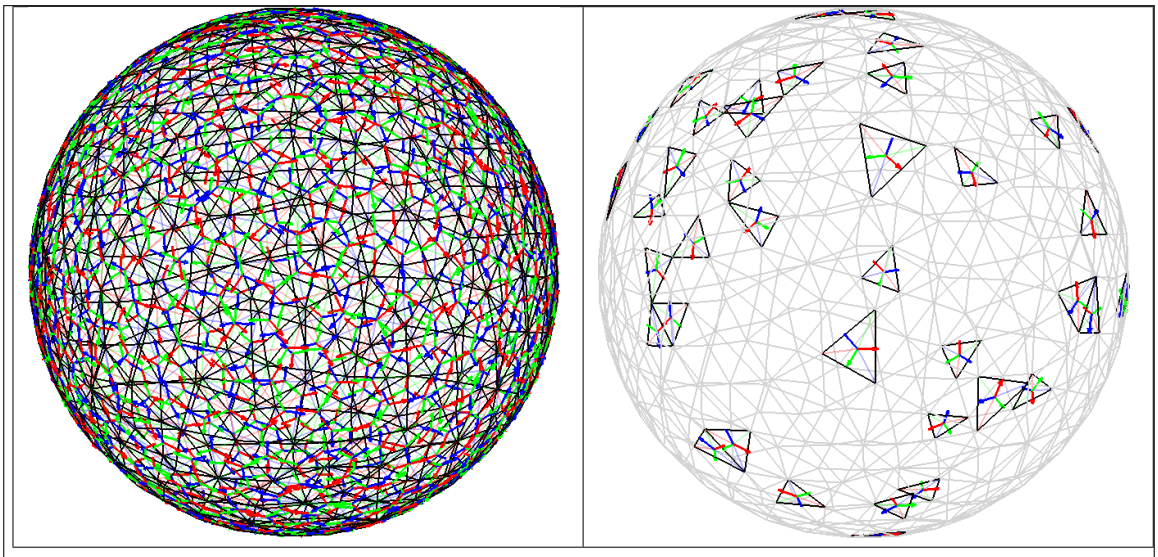
2.6.2 3D example



```
q3,me3=getMesh3D(3,small=True)[:2]
q2,me2=getMesh3D(2,small=True)[:2]
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray')
mlab4sim.plot_normal_faces(q3,me3)
Ba=mshsim.barycenters(q3,me3)
G=mshsim.GradBaCo(q3,me3)
Normals=mshsim.NormalFaces(q3,me3,G)
indices=np.arange(0,me3.shape[1],20)
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray')
mlab4sim.plot_normal_faces(q3,me3,indices=indices,visible_enabled=True,barycenters=Ba,normals=Normals)
```

Listing 17: `plot_normal_faces` function on a 3D example

2.6.3 3D surface example



```
q2,me2=getMesh3Ds(2,small=True)[:2]
q1,me1=getMesh3Ds(1,small=True)[:2]
mlab.figure(1,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='k')
mlab4sim.plot_normal_faces(q2,me2)
Ba=mshsim.barycenters(q2,me2)
G=mshsim.GradBaCo(q2,me2)
Normals=mshsim.NormalFaces(q2,me2,G)
indices=np.arange(0,me2.shape[1],20)
mlab.figure(2,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray')
mlab4sim.plot_normal_faces(q2,me2,indices=indices,visible_enabled=True,barycenters=Ba,normals=Normals)
```

Listing 18: `plot_normal_faces` function on 3D surface example

3 Data functions

3.1 function `plot`

The PLOT function displays scalar datas on a mesh given by a vertices array `q` and a connectivity array `me`.

Syntaxe

```
obj=mlab4sim.plot(q,me,u)
obj=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

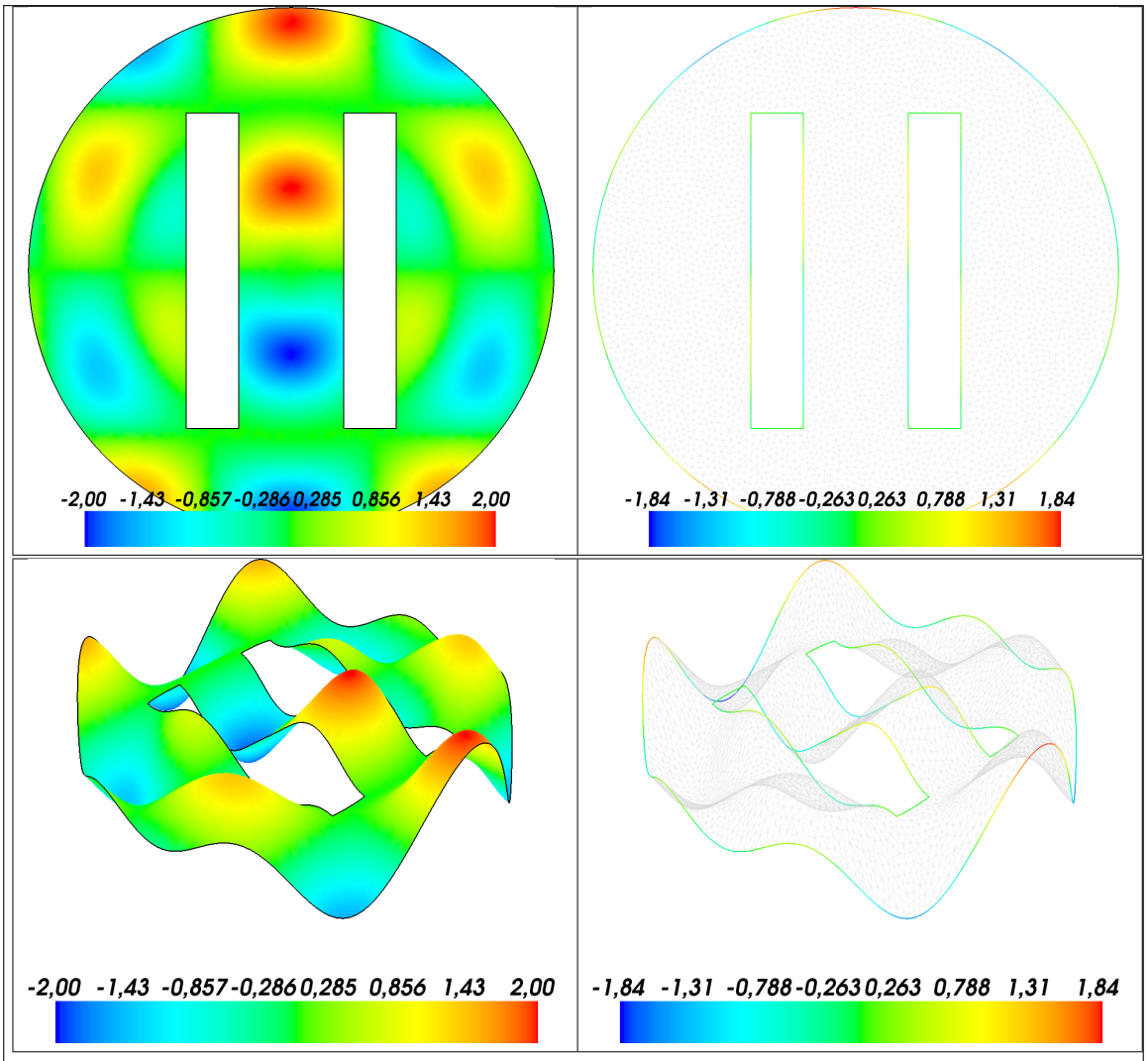
`mlab4sim.plot(q,me,u,Key=Value, ...)` specifies function options using one or more `Key`,`Value` pair arguments. Options of first level are

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

The second level options are those of the `mlab.pipeline.surface` function

The `obj` output is the graphical object created by the function.

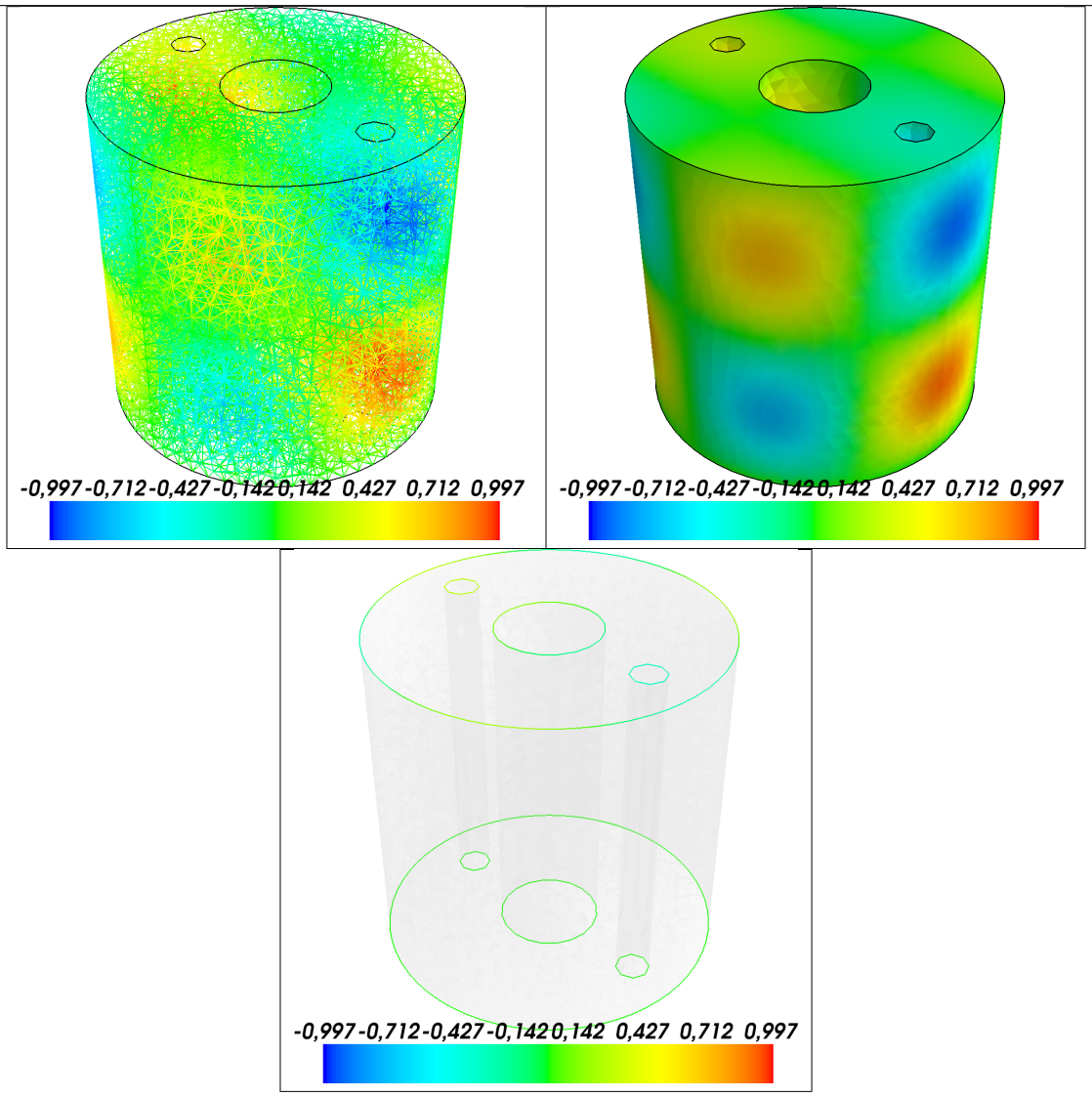
3.1.1 2D example




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

Listing 19: 2D mesh : `plot` function

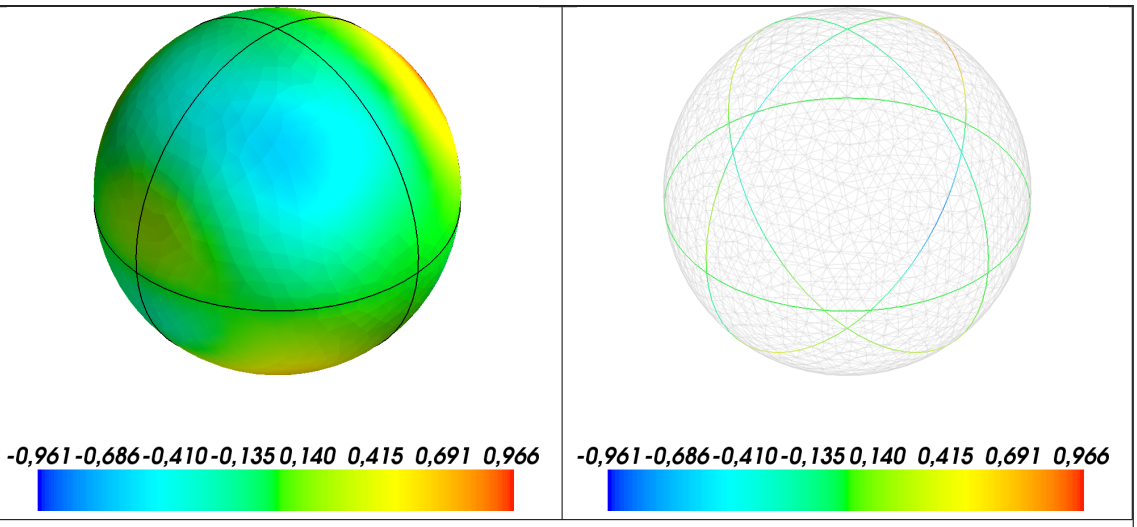
3.1.2 3D example




```
q3,me3=getMesh3D(3)[:2]
q2,me2=getMesh3D(2)[:2]
q1,me1=getMesh3D(1)[:2]
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.figure(1,**fig_opts)
mlab4sim.plot(q3,me3,u3)
mlab4sim.plotmesh(q1,me1,color='Black',line_width=2)
mlab.colorbar()
mlab.figure(2,**fig_opts)
mlab4sim.plot(q2,me2,u2)
mlab4sim.plotmesh(q1,me1,color='Black',line_width=2)
mlab.colorbar()
mlab.figure(3,**fig_opts)
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 20: 3D mesh : `plot` function

3.1.3 3D surface example




```
q2,me2=getMesh3Ds(2)[:2]
q1,me1=getMesh3Ds(1)[:2]
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,**fig_opts)
mlab4sim.plot(q2,me2,u2)
mlab4sim.plotmesh(q1,me1,color='Black',line_width=2)
mlab.colorbar()
mlab.figure(2,**fig_opts)
mlab4sim.plot(q1,me1,u1,line_width=2,vmin=min(u2),vmax=max(u2))
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.2)
mlab.colorbar()
```

Listing 21: 3D surface mesh : `plot` function

3.2 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

```
obj=mlab4sim.plotiso(q,me,u)
obj=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`.

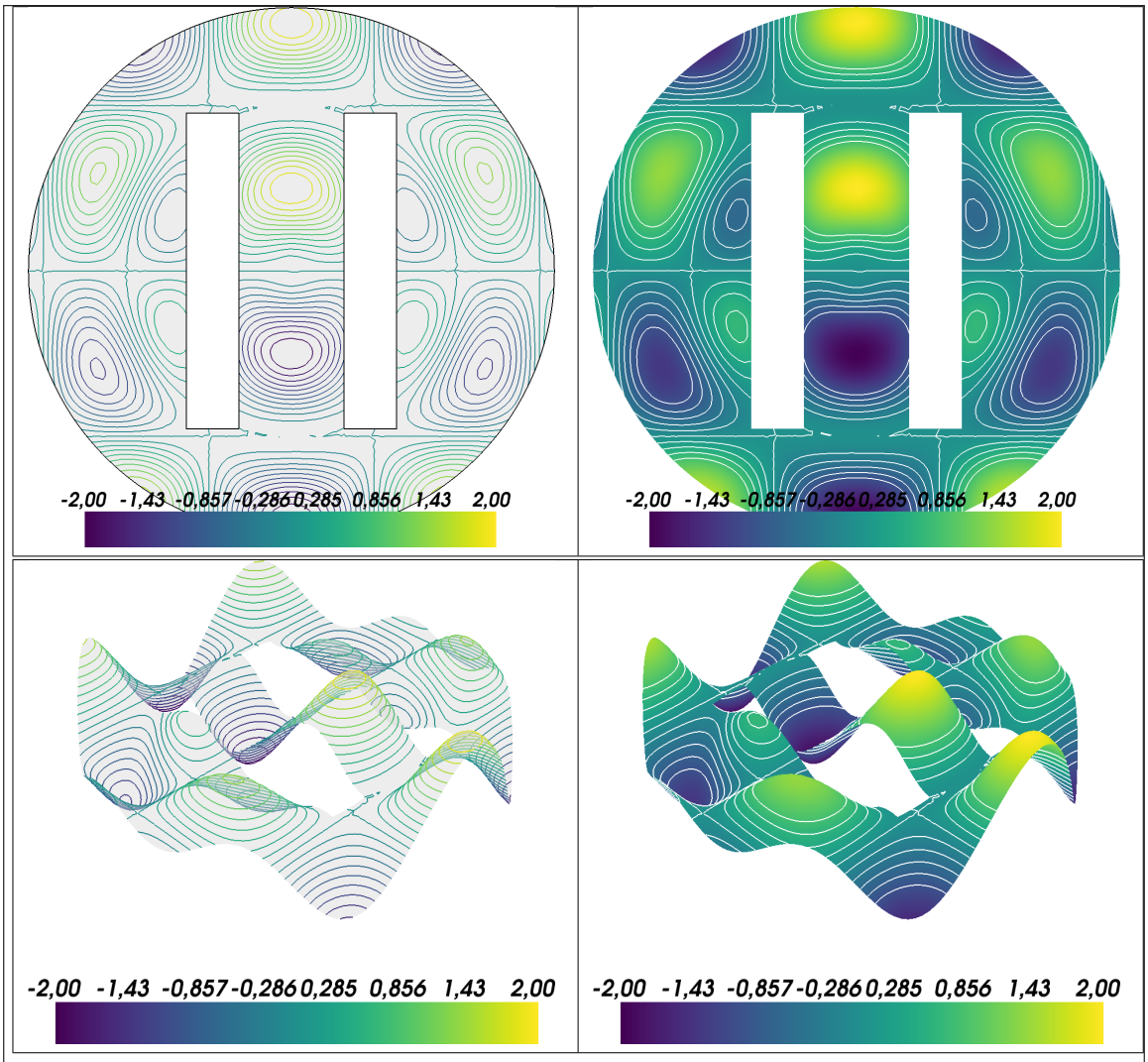
`mlab4sim.plotiso(q,me,u,key=value, ...)` specifies function options using one or more `key`,`value` pair arguments. Options of first level are

- `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.

The `obj` output is the graphical object created by the function.

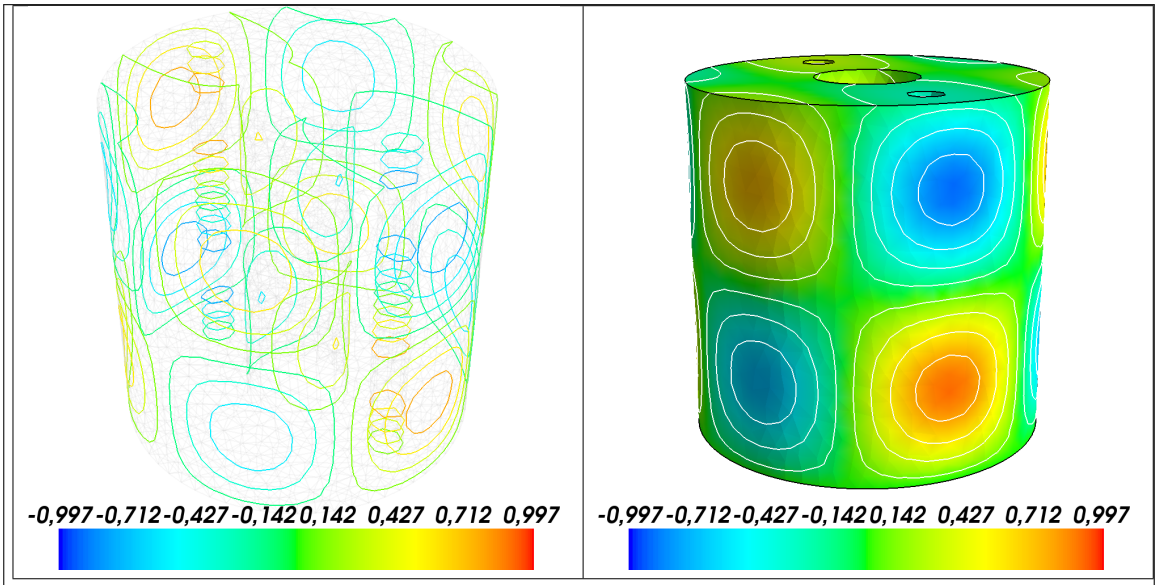
3.2.1 2D example




```
q2,me2=getMesh2D(2)[:2]
q1,me1=getMesh2D(1)[:2]
f=lambda x,y: 2*np.sin(np.sqrt(x**2+y**2))*np.cos(x)*np.sin(y)
u2=f(q2[0],q2[1])
mlab.close(all=True)
mlab.figure(1,**fig_opts)
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,**fig_opts)
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,**fig_opts)
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,**fig_opts)
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 22: 2D mesh : plotiso function

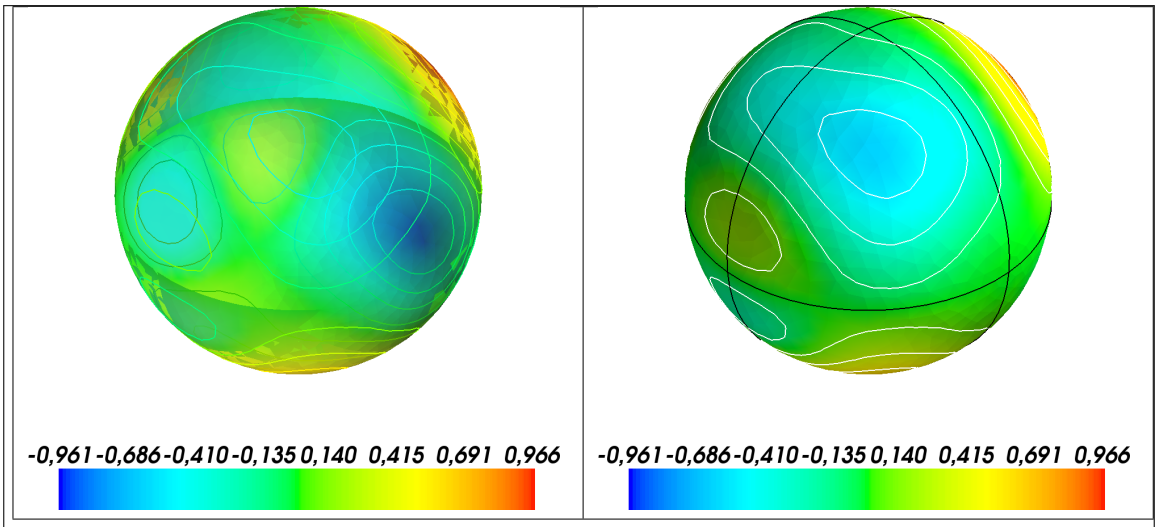
3.2.2 3D example




```
q3,me3=getMesh3D(3)[:2]
q2,me2=getMesh3D(2)[:2]
q1,me1=getMesh3D(1)[:2]
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,**fig_opts)
mlab4sim.plotiso(q2,me2,u2,line_width=2)
mlab4sim.plotmesh(q2,me2, color='LightGray', opacity=0.1)
mlab.colorbar()
mlab.figure(2,**fig_opts)
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 23: 3D mesh : `plotiso` function

3.2.3 3D surface example




```
q2,me2=getMesh3Ds(2)[:2]
q1,me1=getMesh3Ds(1)[:2]
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,**fig_opts)
mlab4sim.plot(q2,me2,u2, opacity=0.7)
mlab4sim.plotiso(q2,me2,u2, line_width=1.5)
mlab.colorbar()
mlab.figure(2,**fig_opts)
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 24: 3D surface mesh : `plotiso` function

3.3 function `quiver`

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

Syntaxe

```
obj=mlab4sim.quiver(q,me,V)
obj=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` .

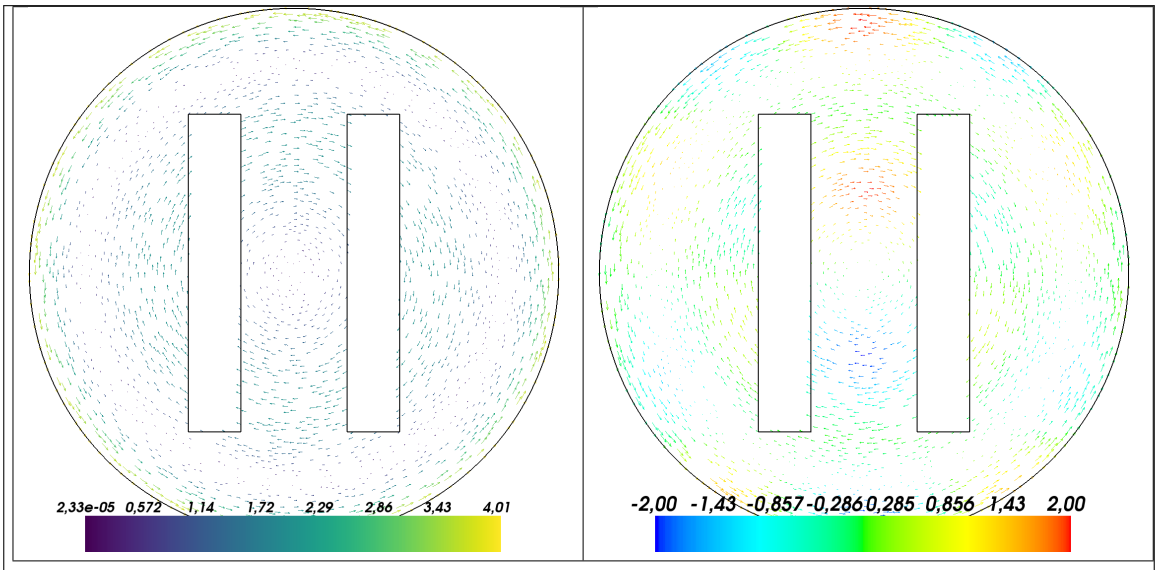
`mlab4sim.quiver(q,me,V,Key=Value, ...)` specifies function options using one or more `Key`,`Value` pair arguments. Options of first level are

- `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.

The `obj` output is the graphical object created by the function.

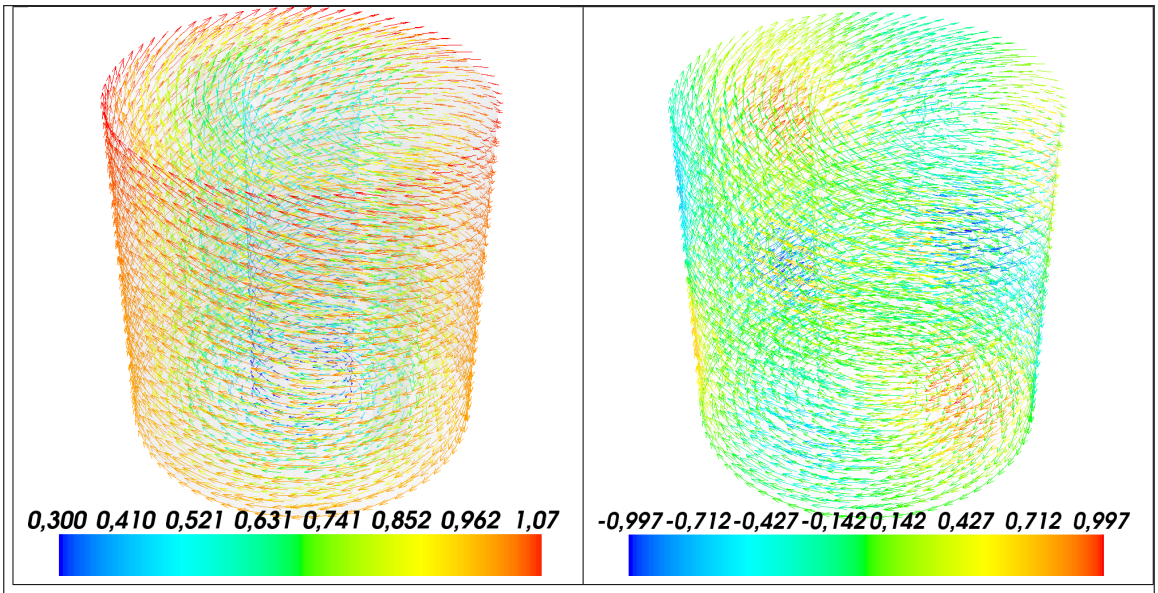
3.3.1 2D example




```
q2,me2=getMesh2D(2)[:2]
q1,me1=getMesh2D(1)[:2]
f=lambda x,y: 2*np.sin(np.sqrt(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])])
nq=q2.shape[1]
mlab.figure(1,**fig_opts)
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,**fig_opts)
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 25: 2D mesh : `quiver` function

3.3.2 3D example

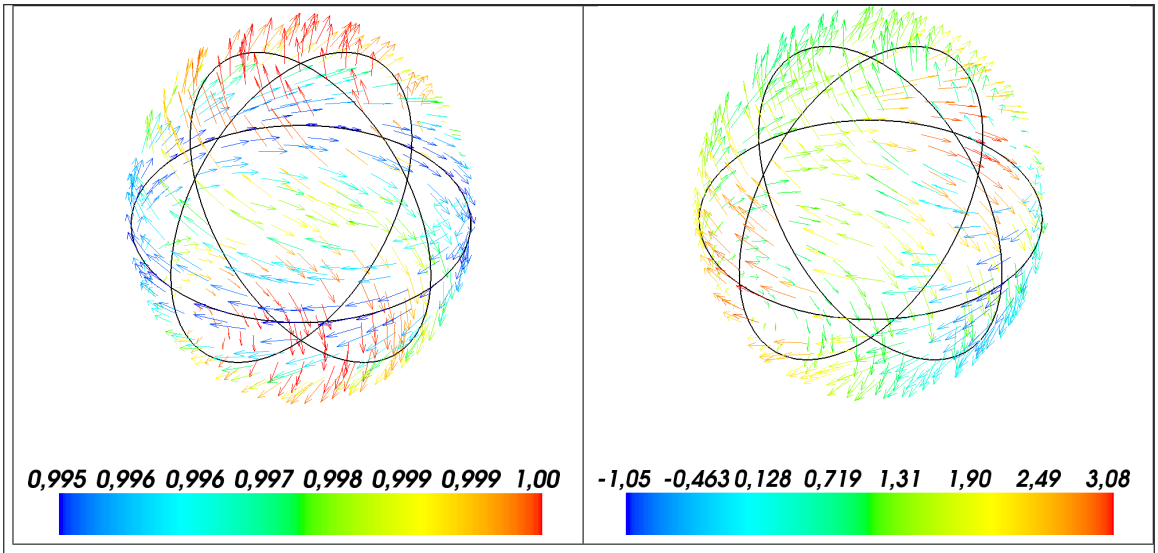



```
q3,me3=getMesh3D(3)[:2]
q2,me2=getMesh3D(2)[:2]
q1,me1=getMesh3D(1)[:2]
f=lambda x,y,z: np.cos(3*x)*np.sin(2*y)*np.sin(3*z)
u=f(q3[0],q3[1],q3[2])
w=[lambda x,y,z: y*np.cos(-(x**2+y**2)/10), lambda x,y,z: -x*np.cos(-(x**2+y**2)/10), lambda
   x,y,z: z/5]
W=np.array([w[0](q3[0],q3[1],q3[2]),
            w[1](q3[0],q3[1],q3[2]),
            w[2](q3[0],q3[1],q3[2])])
nq=q3.shape[1]
mlab.figure(1,**fig_opts)
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,**fig_opts)
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 26: 3D mesh : quiver function

3.3.3 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)[:2]
q1,me1=getMesh3Ds(1)[:2]
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])])

nq=q2.shape[1]
mlab.figure(1,**fig_opts)
mlab4sim.quiver(q2,me2,W,line_width=1,mask_points=max(1,nq//400))
mlab4sim.plotmesh(q1,me1,color='black',line_width=2)
mlab.vectorbar()
mlab.figure(2,**fig_opts)
mlab4sim.quiver(q2,me2,W,scalars=u, line_width=1,mask_points=max(1,nq//400))
mlab4sim.plotmesh(q1,me1,color='black',line_width=2)
mlab.scalarbar()
```

Listing 27: 3D surface mesh : `quiver` function

3.4 function `slice`

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

Syntaxe

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

Description

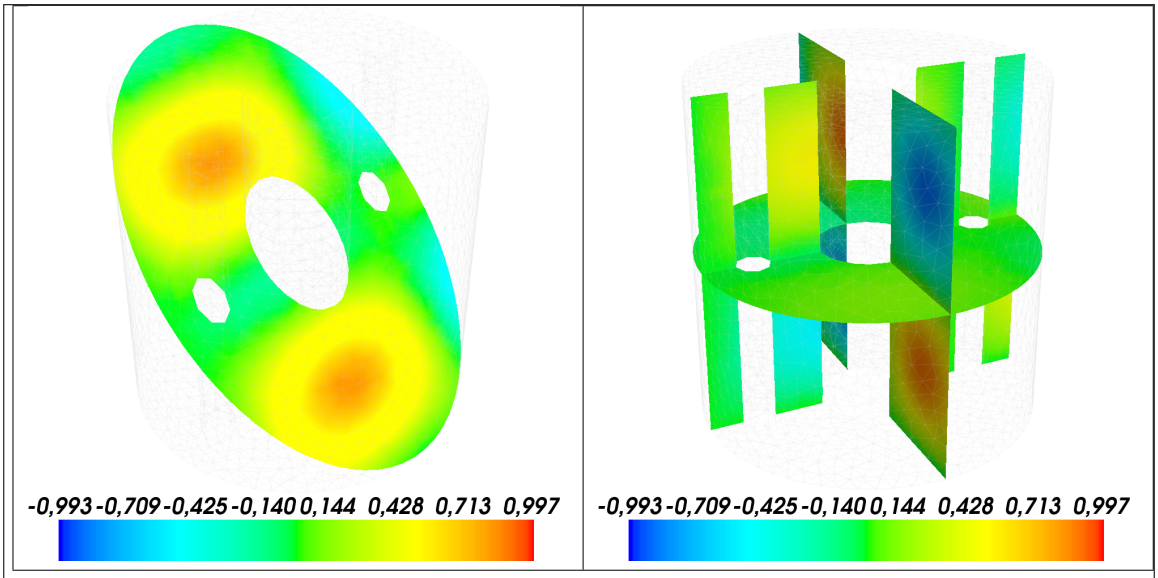
`mlab4sim.slice(q,me,u)` displays `u` data on the intersection of a plane and all the 3-dimensional simplices elements. By default the plane is given by an *origin* point $(0,0,0)$ which lies on it and a *normal* vector $(0,0,1)$ which is orthogonal to the plane. The data `u` is an 1D-array of size `nq`.

`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`.

The `obj` output is the graphical object created by the function.




```
q3,me3=getMesh3D(3)[:2]
q2,me2=getMesh3D(2)[:2]
q1,me1=getMesh3D(1)[:2]
f=lambda x,y,z: np.cos(3*x-1)*np.sin(2*y-2)*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.figure(1,**fig_opts)
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,**fig_opts)
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 28: 3D mesh : `slice` function

3.5 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

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

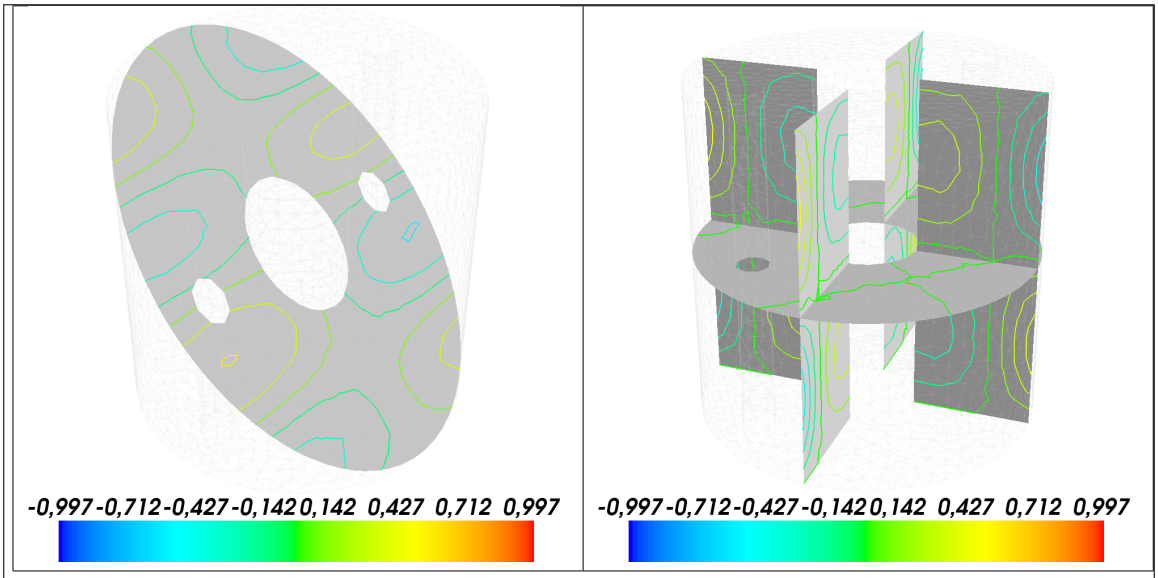
For `key/value` pairs, one could also used those of the `iso_surface` function from `mayavi.mlab.pipeline`.

The `obj` output is the graphical object created by the function.

remark 3.1

This function has a bug: sometines a *core dumped* occurs!

```
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc
Abandon (core dumped)
```




```
q3,me3=getMesh3D(3)[:2]
q2,me2=getMesh3D(2)[:2]
q1,me1=getMesh3D(1)[:2]
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,**fig_opts)
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,**fig_opts)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
# for normal in [(1,0,0),(0,1,0),(0,0,1)]: # BUG: core dumped with this loop
for normal in [(0,0,1),(1,-1,0),(1,1,0)]:
  mlab4sim.slicemesh(q3,me3,origin=(0,0,1),normal=normal, color='LightGray')
  mlab4sim.sliceiso(q3,me3,u3,origin=(0,0,1),normal=normal,contours=15)
mlab.view(155,66,7)
mlab.colorbar()
```

Listing 29: 3D mesh : `sliceiso` function

3.6 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

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

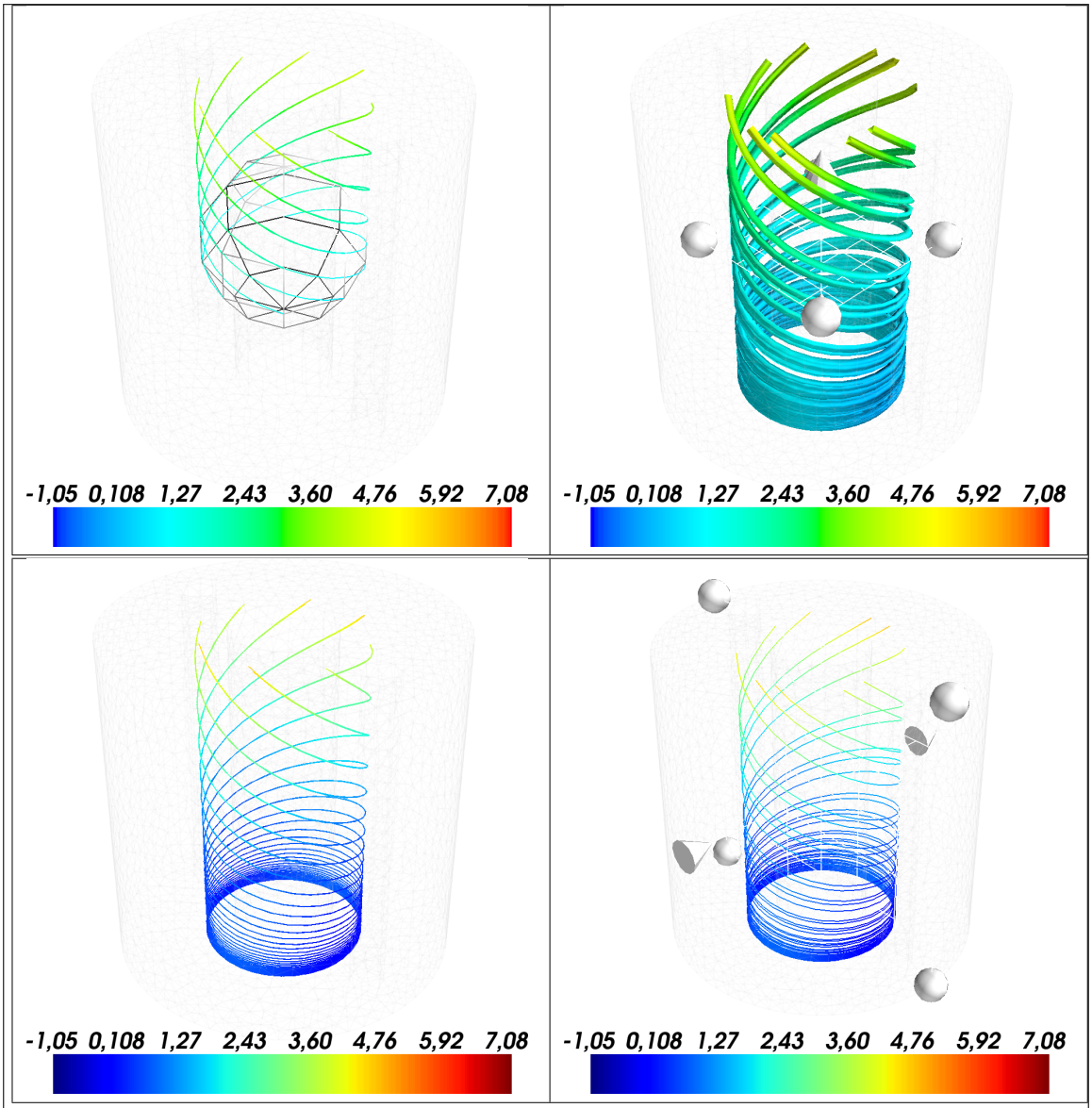
`mlab4sim.streamline(q,me,u,V,Key=Value, ...)` specifies function options using one or more `Key`,`Value` pair arguments. `Key` could be:

- `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 `obj` output is the graphical object created by the function.

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)[:2]
q2,me2=getMesh3D(2)[:2]
q1,me1=getMesh3D(1)[:2]
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,**fig_opts)
mlab4sim.streamline(q3,me3,u,W)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
mlab.colorbar()
mlab.figure(2,**fig_opts)
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,**fig_opts)
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,**fig_opts)
sw_options={'origin':(0,-1,0),'point1':(0,-1,2),'point2':(0,1,0),
            'enabled':True,'resolution':6}
```

3.7 function vectors

The VECTORS function displays vector field by using mlab.pipeline.vectors function on a mesh given by a vertices and connectivity arrays.

Syntaxe

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

Description

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

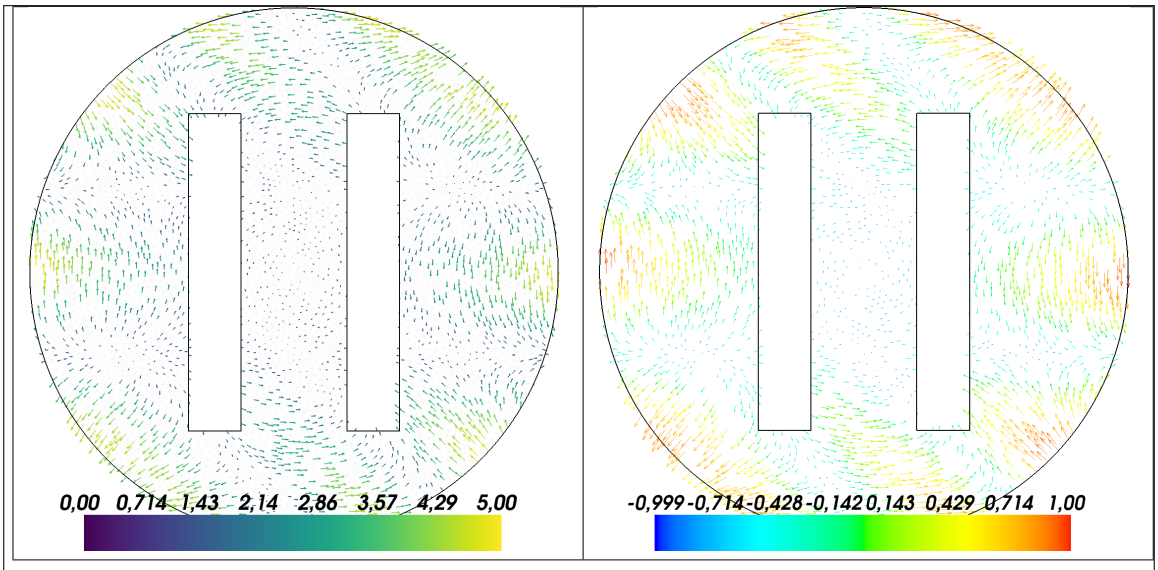
- scalars : specifies a scalar data using as colors
- scalars_name :
- vectors_name :

Options of second level are those of the mlab.pipeline.vectors function:

- color: the color of the vtk object. Overides the colormap, if any, when specified. This is specified as a triplet of float ranging from 0 to 1, eg (1, 1, 1) for white.
- colormap: type of colormap to use.
- extent: [xmin, xmax, ymin, ymax, zmin, zmax] Default is the x, y, z arrays extent. Use this to change the extent of the object created.
- figure: Must be a Scene or None.
- line_width: The width of the lines, if any used. Must be a float. Default: 2.0
- mask_points: If supplied, only one out of 'mask_points' data point is displayed. This option is useful to reduce the number of points displayed on large datasets Must be an integer (int or long) or None.
- mode: the mode of the glyphs. Must be '2darrow' or '2dcircle' or '2dcross' or '2ddash' or '2ddiamond' or '2dhooked_arrow' or '2dsquare' or '2dthick_arrow' or '2dthick_cross' or '2dtriangle' or '2dvertex' or 'arrow' or 'axes' or 'cone' or 'cube' or 'cylinder' or 'point' or 'sphere'. Default: '2darrow'
- name: the name of the vtk object created.
- opacity: The overall opacity of the vtk object. Must be a float. Default: 1.0.
- reset_zoom: Reset the zoom to accomodate the data newly added to the scene. Defaults to True.
- resolution: The resolution of the glyph created. For spheres, for instance, this is the number of divisions along theta and phi. Must be an integer (int or long). Default: 8
- scale_factor: the scaling applied to the glyphs. The size of the glyph is by default in drawing units. Must be a float. Default: 1.0
- scale_mode: the scaling mode for the glyphs ('vector', 'scalar', or 'none').
- transparent: make the opacity of the actor depend on the scalar.
- vmax: vmax is used to scale the colormap. If None, the max of the data will be used
- vmin: vmin is used to scale the colormap. If None, the min of the data will be used

The obj output is the graphical object created by the function.

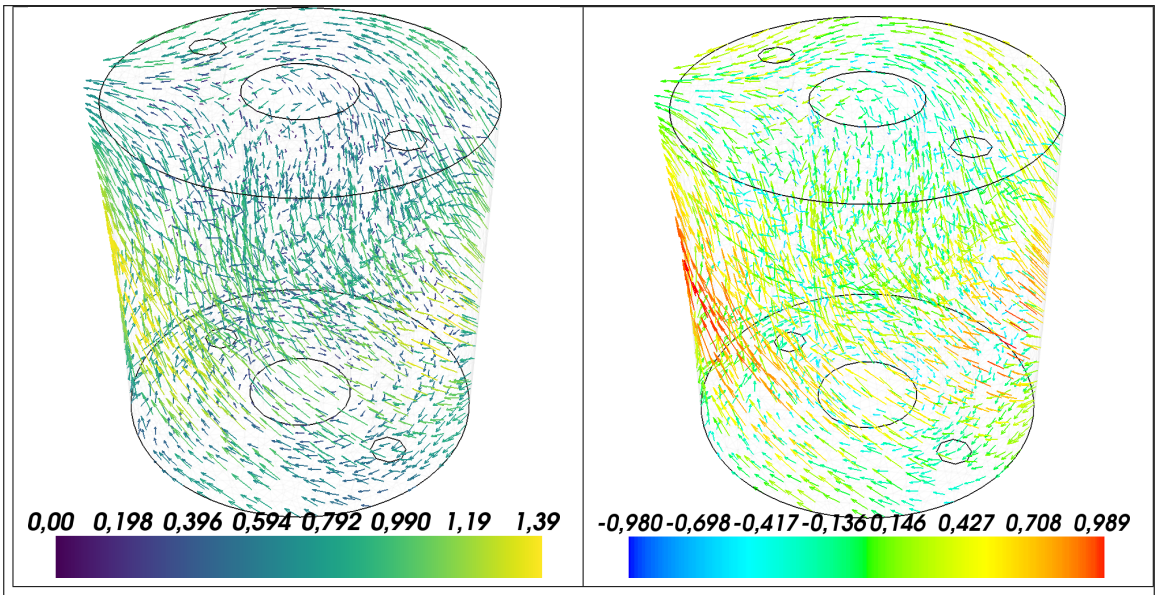
3.7.1 2D example




```
q2,me2=getMesh2D(2)[:2]
q1,me1=getMesh2D(1)[:2]
f=lambda x,y: np.cos(3*x-y)*np.sin(2*y+x)
u2=f(q2[0],q2[1])
w=[lambda x,y: y*np.cos(x+y), lambda x,y: -x*np.cos(y)]
W=np.array([w[0](q2[0],q2[1]),
            w[1](q2[0],q2[1])])
nq=q2.shape[1]
mlab.figure(1,**fig_opts)
mv1=mlab4sim.vectors(q2,me2,W, mask_points=2, scale_factor=0.2,colormap='viridis')
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
mlab4sim.plotmesh(q1,me1,color='k')
mlab.vectorbar()
mlab.view(0,0)
mlab.figure(2,**fig_opts)
mlab4sim.vectors(q2,me2,W, scalars=u2, scale_factor=0.3,line_width=1, mask_points=nq//2000)
mlab4sim.plotmesh(q1,me1,color='black')
mlab.scalarbar()
mlab.view(0,0)
```

Listing 31: 2D mesh : `vectors` function

3.7.2 3D example



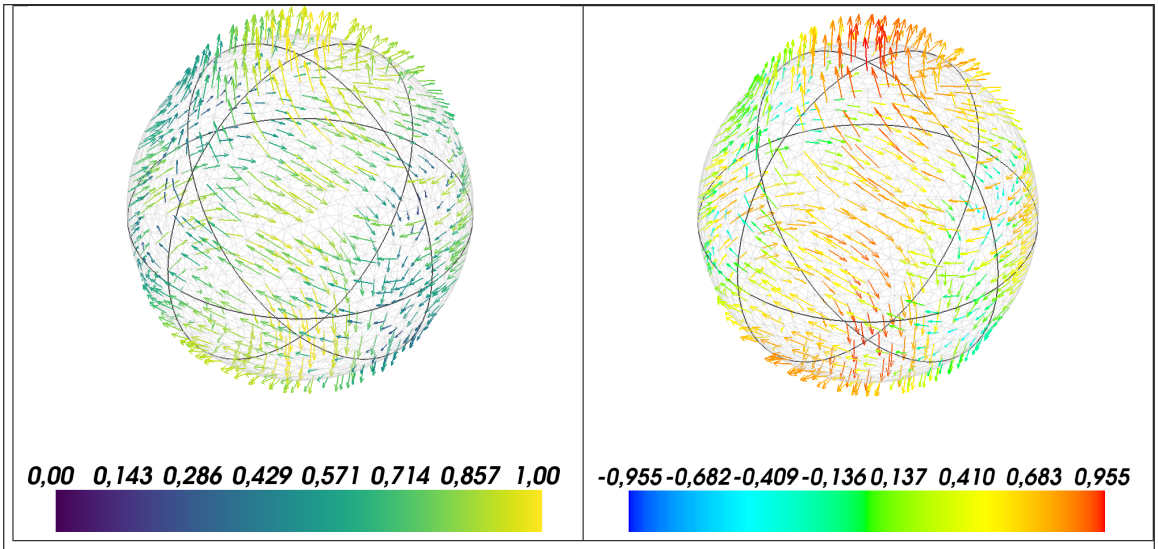

```
q3,me3=getMesh3D(3)[:2]
q2,me2=getMesh3D(2)[:2]
q1,me1=getMesh3D(1)[:2]
f=lambda x,y,z: np.cos(3*x-y)*np.sin(2*y+x)*np.sin(3*z)
u3=f(q3[0],q3[1],q3[2])
u2=f(q2[0],q2[1],q2[2])
w=[lambda x,y,z: y*np.cos(x+y-z), lambda x,y,z: -x*np.cos(-x+y+z), lambda x,y,z:
   (2-z)*z*np.sin(x+y+z)]
W=np.array([w[0](q3[0],q3[1],q3[2]),
            w[1](q3[0],q3[1],q3[2]),
            w[2](q3[0],q3[1],q3[2])])
mlab.close(all=True)
mlab.figure(1,**fig_opts)
mlab4sim.vectors(q3,me3,W, mask_points=2, scale_factor=0.2,colormap='viridis')
mlab4sim.plotmesh(q1,me1,color='k')
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
mlab.vectorbar()

mlab.figure(2,**fig_opts)
mlab4sim.vectors(q3,me3,W,scalars=u3, mask_points=2, scale_factor=0.2)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
mlab4sim.plotmesh(q1,me1,color='k')
mlab.scalarbar()
```

Listing 32: 3D mesh : `vectors` function

3.7.3 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)[:2]
q1,me1=getMesh3Ds(1)[:2]
f=lambda x,y,z: np.cos(3*x-y)*np.sin(2*y+x)*np.sin(3*z)
u2=f(q2[0],q2[1],q2[2])
w=[lambda x,y,z: y*np.cos(x+y-z), lambda x,y,z: -x*np.cos(z+x-y), lambda x,y,z: z*np.cos(x-y)]
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.figure(1,**fig_opts)
mlab4sim.vectors(q2,me2,W, mask_points=2, scale_factor=0.2,colormap='viridis')
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.2)
mlab4sim.plotmesh(q1,me1,color='k')
mlab.vectorbar()
mlab.figure(2,**fig_opts)
mlab4sim.vectors(q2,me2,W,scalars=u2, mask_points=2, scale_factor=0.2)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.2)
mlab4sim.plotmesh(q1,me1,color='k')
mlab.scalarbar()
```

Listing 33: 3D surface mesh : `vectors` function

3.8 function `volume`

The VOLUME function visualizes scalar fields using volumetric visualization techniques by using `mlab.pipeline.volume` function on a mesh given by a vertices array `q` and a connectivity array `me`.

Syntaxe

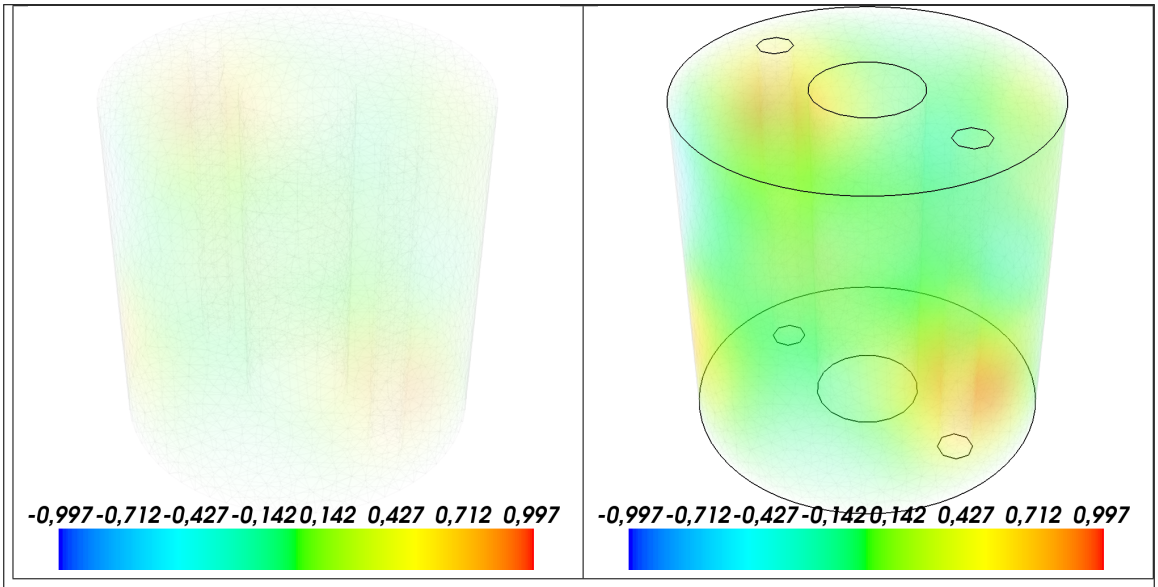
```
obj=mlab4sim.volume(q,me,u)
obj=mlab4sim.volume(q,me,u,Key=Value, ...)
```

Description

`mlab4sim.volume(q,me,u)` visualizes scalar field `u` using volumetric visualization techniques and returns the graphical object created.

`mlab4sim.volume(q,me,u,Key=Value, ...)` specifies function options using one or more `Key`,`Value` pair arguments. They are those of the `volume` function from `mayavi.mlab.pipeline`.

The `obj` output is the graphical object created by the function.




```
q3,me3=getMesh3D(3)[:2]
q2,me2=getMesh3D(2)[:2]
q1,me1=getMesh3D(1)[:2]
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])
mlab.figure(1,**fig_opts)
mlab4sim.volume(q3,me3,u)
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
mlab.colorbar()
mlab.figure(2,**fig_opts)
mvol=mlab4sim.volume(q3,me3,u)
mvol._volume_property.scalar_opacity_unit_distance=0.2
mlab4sim.plotmesh(q2,me2,color='LightGray',opacity=0.05)
mlab4sim.plotmesh(q1,me1,color='k')
mlab.colorbar()
```

Listing 34: 3D mesh : `volume` function

3.9 function `slice_unstructured_grid`

The `slice_unstructured_grid` function displays the mesh elements intersecting a slice (plane) by using `mlab.pipeline.slice_unstructured_grid` function.

Syntaxe

```
obj=mlab4sim.slice_unstructured_grid(*args,**kwargs)
```

Description

`obj=mlab4sim.slice_unstructured_grid(q,me)` visualizes mesh elements intersecting the slice.

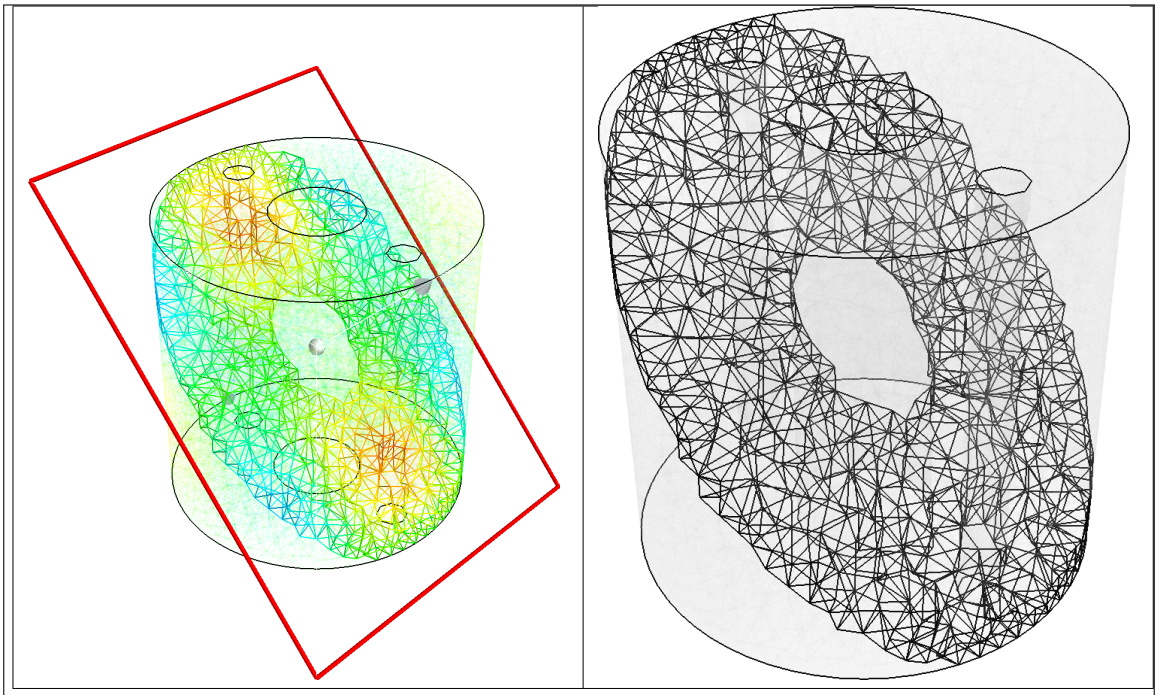
`obj=mlab4sim.slice_unstructured_grid(q,me,u)` visualizes mesh elements intersecting the slice colored by the scalar data `u`, a numpy array of length `nq`.

`obj=mlab4sim.slice_unstructured_grid(obj)` uses the graphical object `obj` to create the slice. The graphical object must contains an unstructured grid. If this unstructured grid contains a scalar data then it is used to color the slice.

`obj=mlab4sim.slice_unstructured_grid(*args,Key=Value, ...)` specifies function options using one or more `Key,Value` pair arguments.

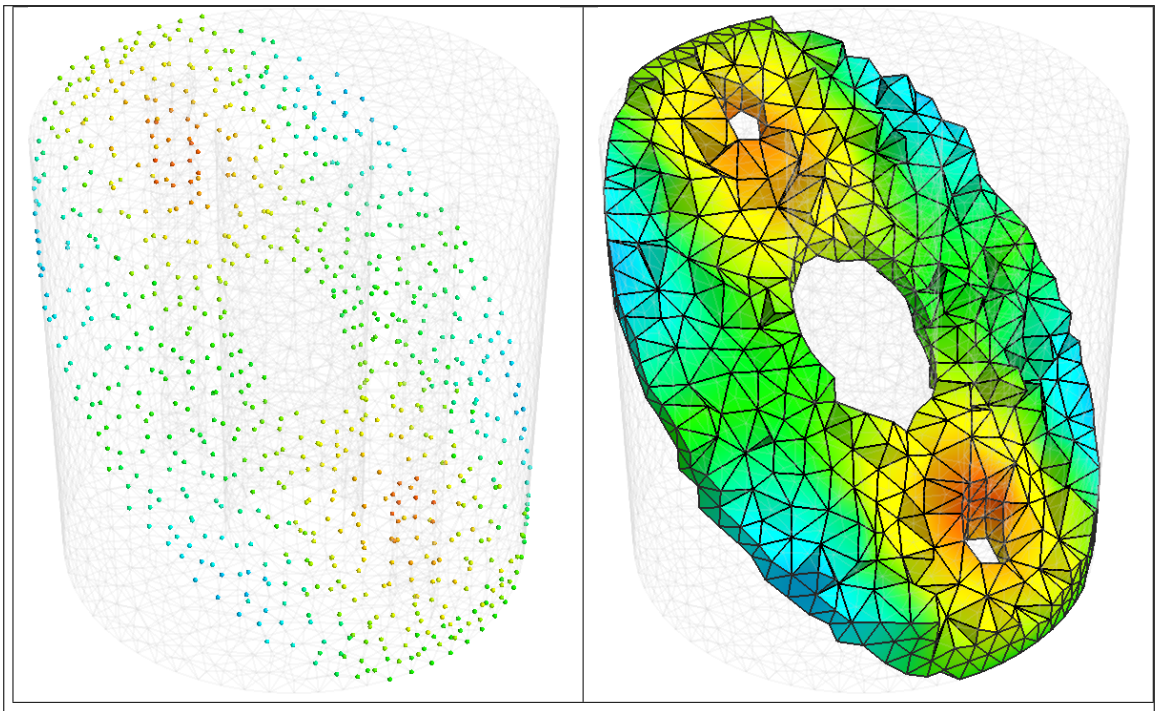
- `origin` : to specify a point lying on the plane (default is the *center* of box containing the mesh)
- `normal` : to specify a vector orthogonal to the plane (default is $(0, 0, 1)$)
- `property` : a dictionary specifying the `actor.property` of the object created by the function `mlab.pipeline.slice_unstructured_grid`. There is a list of some usual keys :

- 'opacity'
- 'representation' controls the surface geometry representation for the object; 'wireframe', 'surface' or 'points'
- 'color'
- 'edge_visibility'
- 'edge_color'
- 'line_width'
- 'point_size'
- 'render_lines_as_tubes'
- 'render_points_as_spheres'



```
q3,me3=getMesh3D(3)[:2]
q2,me2=getMesh3D(2)[:2]
q1,me1=getMesh3D(1)[:2]
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])
mlab.figure(1,**fig_opts)
mp=mlab4sim.plot(q3,me3,u,opacity=0.05)
sug=mlab4sim.slice_unstructured_grid(mp,normal=[0,1,1],origin=[0,0,1],w_enabled=True)
mlab4sim.plotmesh(q1,me1,color='Black',line_width=2)
mlab.figure(2,**fig_opts)
mpm=mlab4sim.plotmesh(q3,me3,opacity=0.05,color='LightGray')
sug=mlab4sim.slice_unstructured_grid(mpm,normal=[0,1,1],origin=[0,0,1], property={'color':'k'})
mlab4sim.plotmesh(q1,me1,color='Black',line_width=2)
mlab.figure(3,**fig_opts)
mpm=mlab4sim.plotmesh(q2,me2,opacity=0.1,color='LightGray')
sug=mlab4sim.slice_unstructured_grid(q3,me3,u,normal=[0,1,1],origin=[0,0,1],
                                     property={ 'representation':'points',
                                                'point_size':5,
                                                'render_points_as_spheres':True}
                                     )
mlab.figure(4,**fig_opts)
mpm=mlab4sim.plotmesh(q2,me2,opacity=0.1,color='LightGray')
sug=mlab4sim.slice_unstructured_grid(q3,me3,u,normal=[0,1,1],origin=[0,0,1],
                                     property={ 'representation':'surface',
                                                'edge_visibility':True,
                                                'line_width':2,
                                                'edge_color':'k'}
                                     )
```

Listing 35: 3D mesh : slice_unstructured_grid01 function, example 1



```
q3,me3=getMesh3D(3)[:2]
q2,me2=getMesh3D(2)[:2]
q1,me1=getMesh3D(1)[:2]
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])
mlab.figure(1,**fig_opts)
mpm=mlab4sim.plotmesh(q2,me2,opacity=0.1,color='LightGray')
sug=mlab4sim.slice_unstructured_grid(q3,me3,u,normal=[0,1,1],origin=[0,0,1],
                                     property={ 'representation':'points',
                                                'point_size':5,
                                                'render_points_as_spheres':True}
                                     )
mlab.figure(2,**fig_opts)
mpm=mlab4sim.plotmesh(q2,me2,opacity=0.1,color='LightGray')
sug=mlab4sim.slice_unstructured_grid(q3,me3,u,normal=[0,1,1],origin=[0,0,1],
                                     property={ 'representation':'surface',
                                                'edge_visibility':True,
                                                'line_width':2,
                                                'edge_color':'k'}
                                     )
```

Listing 36: 3D mesh : `slice_unstructured_grid02` function, example 1

3 References

[1] P. Ramachandran and G. Varoquaux. Mayavi: 3D Visualization of Scientific Data. *Computing in Science & Engineering*, 13(2):40–51, 2011.