fc-hypermesh Python package, User's Guide * version 0.1.1

Francois Cuvelier

hypermesh Python package, User's Guide*
version 0.1.1

François Cuvelier†

December 30, 2019

Abstract

This object-oriented Python package allows in any dimension d to generate conforming meshes of d-orthotopes by p-order simplices or orthotopes with their m-faces. It was created to show the implementation of the algorithms of [1]. The hypermesh package uses Python objects and is provided with meshes visualization tools for dimension less than or equal to 3.

0 Contents

1 Introduction **2**

2 Installation and uninstall **6**

3 Classes of the package **7**
3.1 Class `EltMesh` . . . 7
3.2 Class `OrthMesh` . . . 7
3.2.1 Constructor . . . 8
3.2.2 Access to `OrthMesh`'s fields . . . 10
3.2.3 `plotmesh` method . . . 11
3.2.4 `plotnodes` method . . . 13
3.2.5 `ploteltsNumber` method . . . 14
3.2.6 `ploteltsNumber` method . . . 15

4 Using the `fc_hypermesh` package **16**
4.1 2d-orthotope meshing by simplices . . . 17
4.2 3d-orthotope meshing by simplices . . . 18
4.3 2d-orthotope meshing by orthotopes . . . 19
4.4 3d-orthotope meshing by orthotopes . . . 20
4.5 Mapping of a 2d-orthotope meshing by simplices . . . 21
4.6 Mapping of a 3d-orthotope meshing by orthotopes . . . 22

*LaTeX manual, revision 0.1.1, compiled with Python 3.8.1, and packages `fc-hypermesh`[0.1.1], `fc-tools`[0.0.24], `fc-bench`[0.2.0],

†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

5 Memory consuming **23**

6 Benchmarks **24**
6.1 Tessellation by orthotopes 25
6.2 Tessellation by simplices 26

1 Introduction

The fc-hypermesh package contains a simple class OrthMesh which permits, in any dimension $\mathrm{d} \geqslant 1$, to obtain conforming mesh of a d-orthotope tessellated with p-order simplices or p-order orthotopes. Corresponding m-faces, $0 \leqslant \mathrm{m} < \mathrm{d}$ of the mesh are also provided. The number of m-faces of a d-orthotope is

$$E_m^{\mathrm{d}} \stackrel{\text{def}}{=} 2^{\mathrm{d}-m}\binom{d}{m} \quad \text{where} \quad \binom{\mathrm{d}}{m} = \frac{\mathrm{d}!}{m!(\mathrm{d}-m)!} \tag{1}$$

Results and vectorized algorithms used in this package are given in [1].

For dimension 1 to 3 and order 1 to 4, orthotope elements and simplicial elements are respectively represented In Table 1 and Table 2. In older package(0.0.x versions) only order 1 was provided.

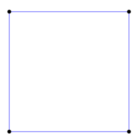
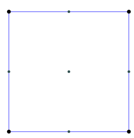
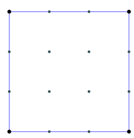
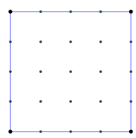
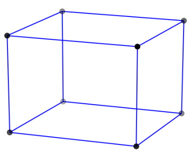
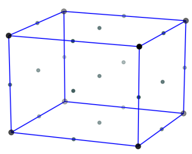
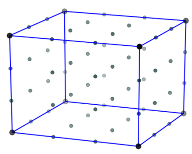
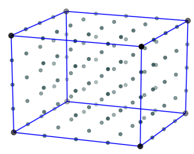
d \ p	1	2	3	4
1				
2				
3				

Table 1: p-order d-orthotope mesh element in $\mathbb{R}^{\mathrm{d}}$. Nodes are the points.

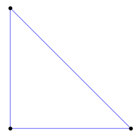
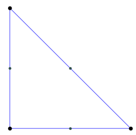
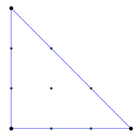
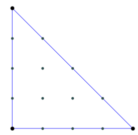
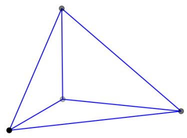
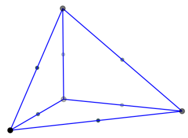
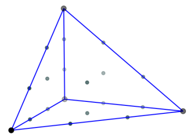
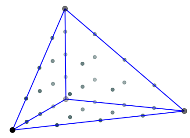
d \ p	1	2	3	4
1				
2				
3				

Table 2: p-order d-simplicial mesh element in $\mathbb{R}^d$. Nodes are the points.

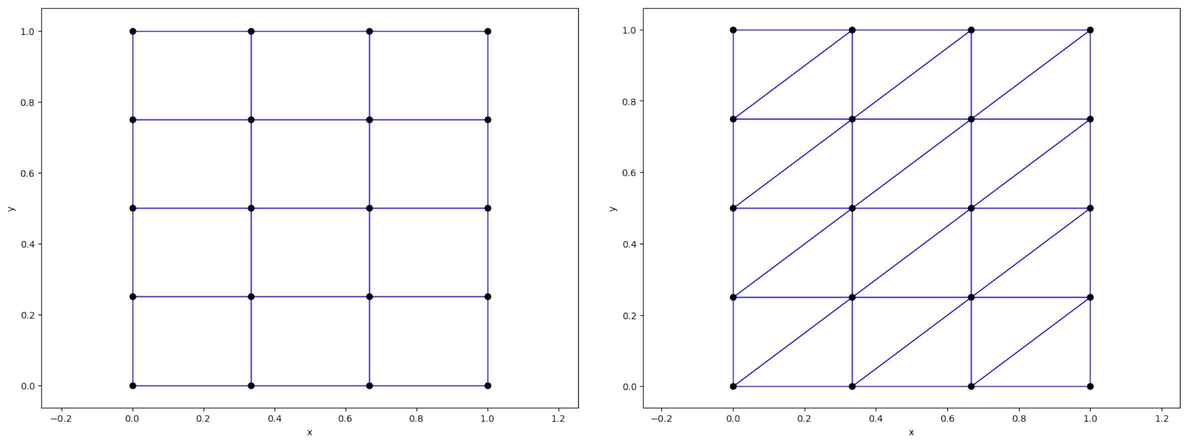


Figure 1: Tesselation samples of $[0,1]^2$ with 1-order 2-orthotopes (left) and 1-order 2-simplices (right) where nodes (vertices) of all mesh elements are represented by black points.

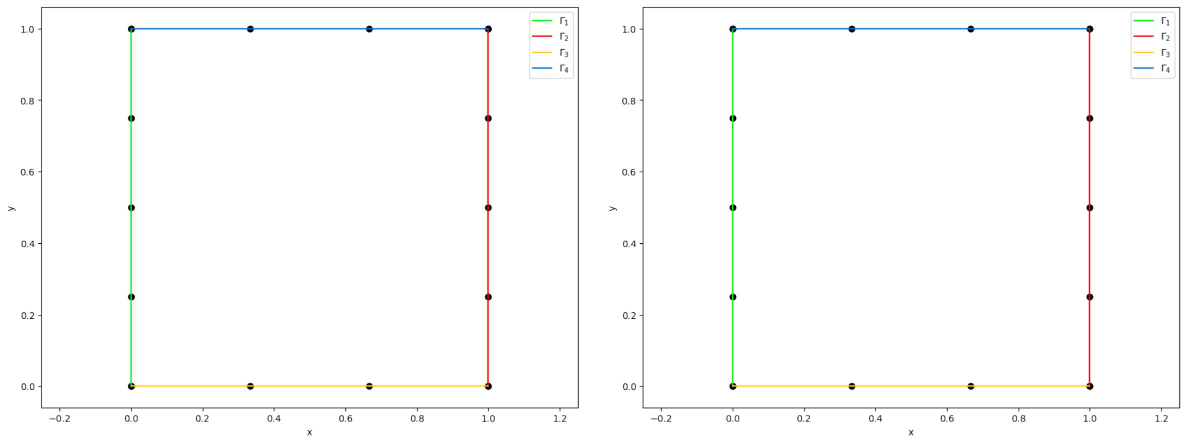


Figure 2: Representation of all the 1-faces meshes with 1-order 1-orthotopes (left) and 1-order 1-simplices (right) obtained from the tesselation samples of the Figure 1

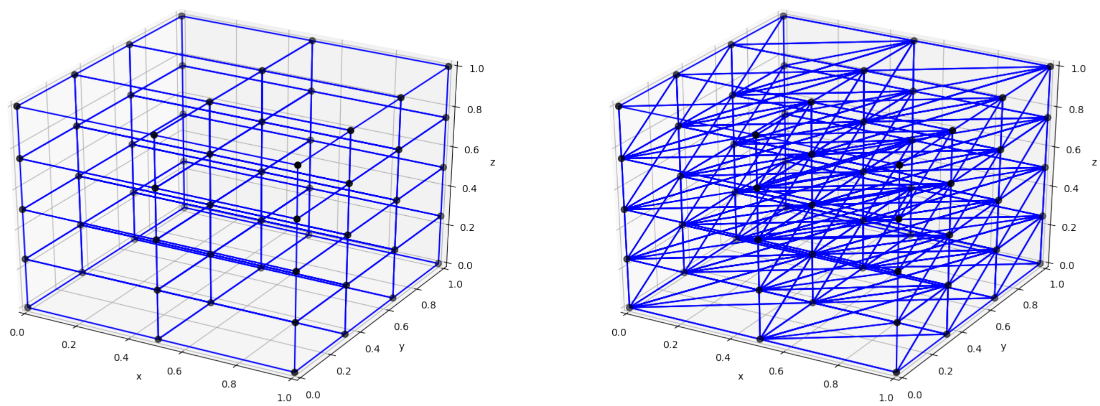


Figure 3: Tesselation samples of $[0, 1]^3$ with 1-order 3-orthotopes (left) and 1-order 3-simplices (right) where nodes (vertices) of all mesh elements are represented by black spheres.

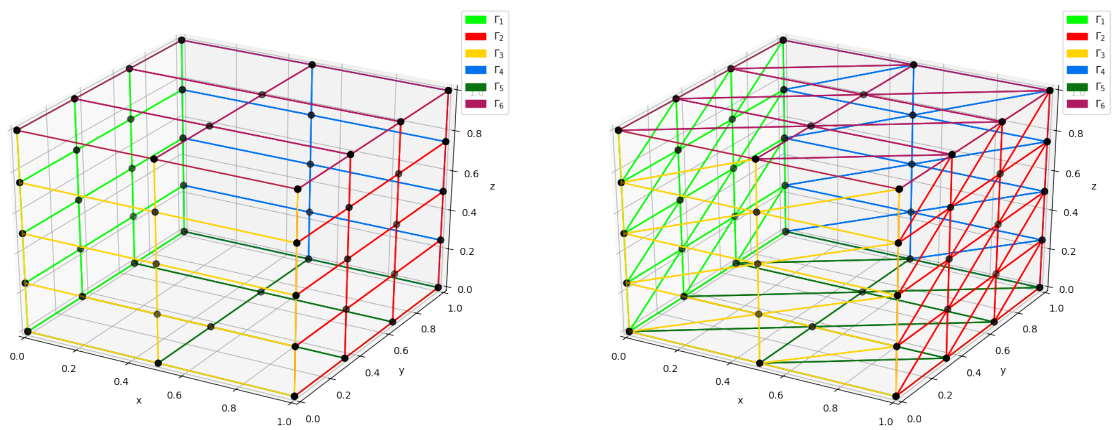


Figure 4: Representation of all the 2-faces meshes with 1-order 2-orthotopes (left) and 1-order 2-simplices (right) obtained from the tesselation samples of the Figure 3

By taking back the meshes in dimension 2 represented in Figure 1 and Figure 2, but this time using 3-order mesh element give the new meshes represented in Figure 5 and Figure 6.

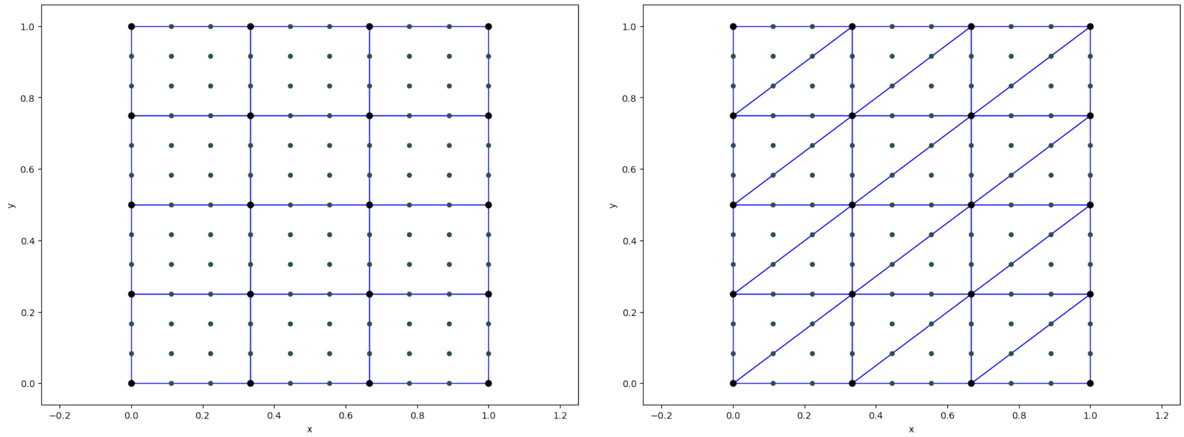


Figure 5: Tesselation samples of $[0, 1]^2$ with 3-order 2-orthotopes (left) and 3-order 2-simplices (right) where nodes of all mesh elements are represented by black (vertices) and grey points.

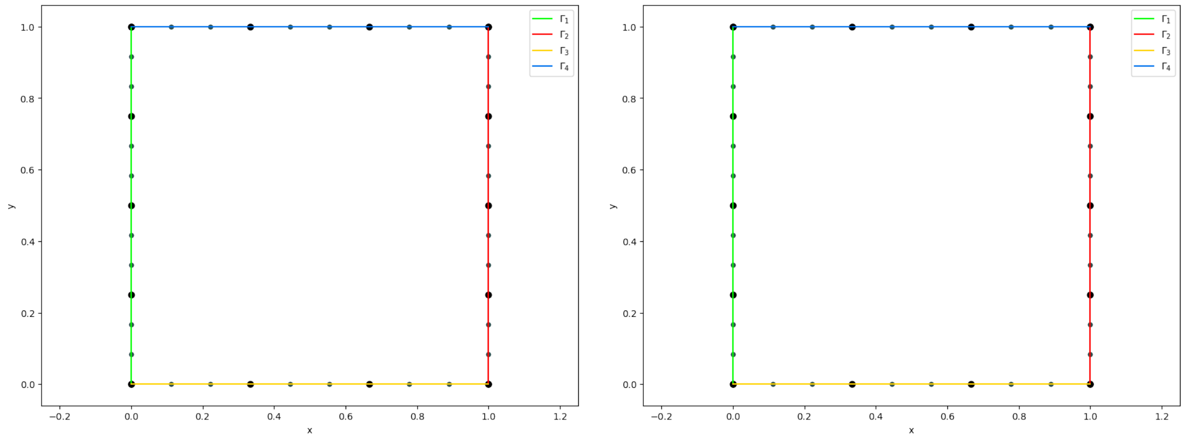


Figure 6: Representation of all the 1-faces meshes with 3-order 1-orthotopes (left) and 3-order 1-simplices (right) obtained from the tesselation samples of the Figure 5

In dimension 3, meshes represented in Figure 3 and Figure 4 are this time tessellated respectively with 3-order orthotopes and 3-order simplices and represented in Figure 7 and Figure 8.

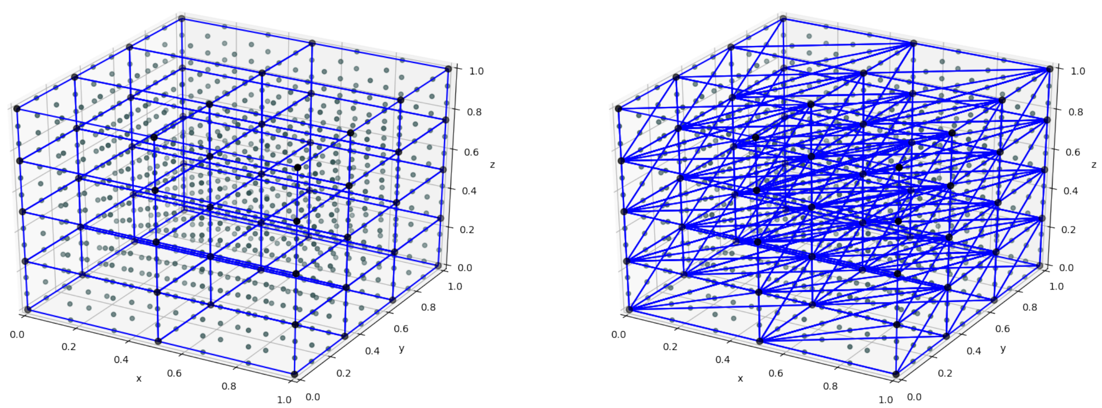


Figure 7: Tesselation samples of $[0, 1]^3$ with 3-order 3-orthotopes (left) and 3-order 3-simplices (right) where nodes of all mesh elements are represented by black (vertices) and grey spheres.

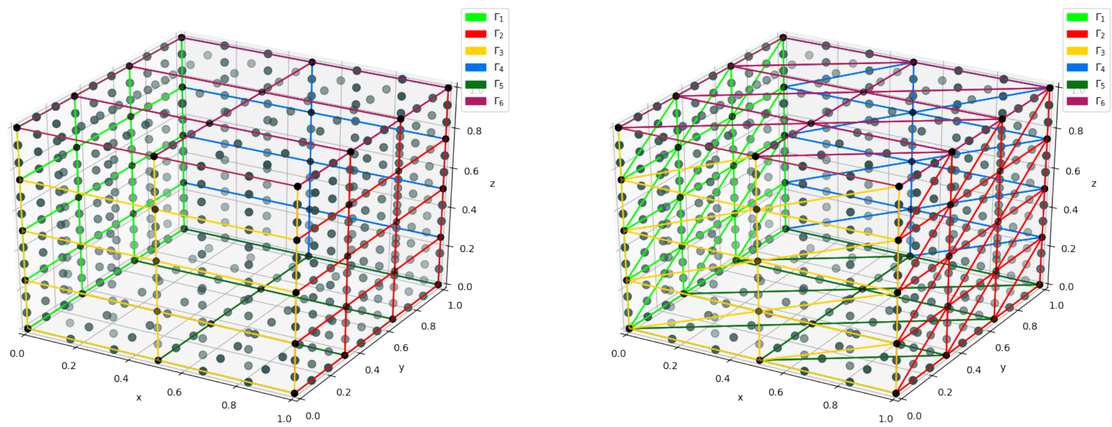


Figure 8: Representation of all the 2-faces meshes with 3-order 2-orthotopes (left) and 3-order 2-simplices (right) obtained from the tesselation samples of the Figure 7

2 Installation and uninstall

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

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

Installation :

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

```
$ pip install -U --user fc_hypermesh
```

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

```
$ sudo pip install -U fc_hypermesh
```

An other way is to download the required archive and to make the installation from the downloaded file.

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

```
$ pip install <PATH_TO_FOLDER>/fc_hypermesh-<VERSION>.tar.gz --user -U
```

  where `<PATH_TO_FOLDER>` will be replaced by the path to the saved archive and `<VERSION>` by the version of the archive.

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

```
$ sudo pip install <PATH_TO_FOLDER>/fc_hypermesh-<VERSION>.tar.gz -U
```

Uninstall : To uninstall this package, you only have to execute one of these commands depending on the type of installation performed

```
$ pip uninstall fc_hypermesh
```

or

```
$ sudo pip uninstall fc_hypermesh
```

3 Classes of the package

First of all, the *low level* class EltMesh is presented. Thereafter the main class OrthMesh, which is an *union* of EltMesh objects, is described.

3.1 Class `EltMesh`

An elementary mesh class EltMesh is used to store only one mesh in space dimension d, the main mesh as well as any of the meshes of the m-faces. This elementary mesh is made either with p-order simplices or with p-order orthotopes. This class EltMesh also simplify (for me) the codes writing. Its attributes are the following:

- d : space dimension
- m : kind of mesh corresponding to a m-face, 0<=m<=d, m == d for the main mesh.
- type : 0 for simplicial mesh or 1 for orthotope mesh.
- order : order p of the elements, default 1.
- nq : number of vertices.
- q : vertices numpy array of dimension d-by-nq
- nme : number of mesh elements
- me : connectivity numpy array of dimension (d+1)-by-nme for simplices elements or 2^d-by-nme for orthotopes elements
- toGlobal : index array linking local array q to the one of the main mesh.
- label : name/number of this elementary mesh
- color : color of this elementary mesh (for plotting purpose)

3.2 Class `OrthMesh`

The aim of the class OrthMesh is to efficiently create an object which contains a main mesh of a d-orthotope and all its m-face meshes deduced from the main mesh. All meshes are made either with p-order simplices or with p-order orthotopes.

Let the d-orthotope defined by $[a_1, b_1] \times \cdots \times [a_{\mathrm{d}}, b_{\mathrm{d}}]$. The class OrthMesh corresponding to this d-orthotope contains the main mesh and all the meshes of its m-faces, $0 \leqslant m < \mathrm{d}$. Its attributes are the following

- d: space dimension.
- type: string 'simplex' or 'orthotope' mesh.
- order: order p of the elements.
- Mesh: main mesh as an EltMesh object.
- Faces: 2d-list of EltMesh objects such that Faces[0] is a list of all the meshes of the $(\mathrm{d}-1)$-faces, Faces[1] is a list of all the meshes of the $(\mathrm{d}-2)$-faces, and so on
- box: a d-by-2 numpy array such that box[i−1][0] is a_i value and box[i−1][1] is b_i value.

In all the Python examples given in the following section, we suppose the following code previously load:

Listing 1: Previously load code for Python examples

```
import fc_hypermesh
import matplotlib.pyplot as plt
from fc_tools.Matplotlib import set_axes_equal
```

3.2.1 Constructor

The OrthMesh constructor is :

Syntaxe

```
Oh = OrthMesh(d,N)
Oh = OrthMesh(d,N,key=value, ...)
```

Description

Oh = OrthMesh(d,N)

Builds an OrthMesh object where N is either a 1-by-d array/list such that N[i−1] is the number of discretization for $[a_i, b_i] = [0, 1]$ or either an integer if the the number of discretization is the same in all space directions. By default, the output OrthMesh object is made with 1-order simplices.

Oh = OrthMesh(d,N,key=value, ...)

Some optional key/value pairs arguments are available with key:

- box : where value is a d-by-2 list or numpy array such that value[i−1][0] is a_i value and value[i−1][1] is b_i value. Default is $[0, 1]^{\mathrm{d}}$.
- type : The default value for optional key parameter type is 'simplex' and ortherwise 'orthotope' can be used

  Listing 2: : `OrthMesh` constructor in dimension `d=3` (orthotope mesh)

  ```
  Oh = fc_hypermesh.OrthMesh(3,10,type='orthotope')
  print(Oh)
  ```

  Output

  ```
  OrthMesh object
        d : 3
     order : 1
       box : [[0.0, 1.0], [0.0, 1.0], [0.0, 1.0]]
   mapping : None
  Mesh (order,type,nq,nme) : (1,orthotope,1331,1000)
  Number of 2-faces : 6
    [ 0] (order,type,nq,nme) : (1,orthotope,121,100)
    [ 1] (order,type,nq,nme) : (1,orthotope,121,100)
    [ 2] (order,type,nq,nme) : (1,orthotope,121,100)
    [ 3] (order,type,nq,nme) : (1,orthotope,121,100)
    [ 4] (order,type,nq,nme) : (1,orthotope,121,100)
    [ 5] (order,type,nq,nme) : (1,orthotope,121,100)
  Number of 1-faces : 12
    [ 0] (order,type,nq,nme) : (1,orthotope,11,10)
    [ 1] (order,type,nq,nme) : (1,orthotope,11,10)
    [ 2] (order,type,nq,nme) : (1,orthotope,11,10)
    [ 3] (order,type,nq,nme) : (1,orthotope,11,10)
    [ 4] (order,type,nq,nme) : (1,orthotope,11,10)
    [ 5] (order,type,nq,nme) : (1,orthotope,11,10)
    [ 6] (order,type,nq,nme) : (1,orthotope,11,10)
    [ 7] (order,type,nq,nme) : (1,orthotope,11,10)
    [ 8] (order,type,nq,nme) : (1,orthotope,11,10)
    [ 9] (order,type,nq,nme) : (1,orthotope,11,10)
    [10] (order,type,nq,nme) : (1,orthotope,11,10)
    [11] (order,type,nq,nme) : (1,orthotope,11,10)
  Number of 0-faces : 8
    [ 0] (order,type,nq,nme) : (1,orthotope,1,1)
    [ 1] (order,type,nq,nme) : (1,orthotope,1,1)
    [ 2] (order,type,nq,nme) : (1,orthotope,1,1)
    [ 3] (order,type,nq,nme) : (1,orthotope,1,1)
    [ 4] (order,type,nq,nme) : (1,orthotope,1,1)
    [ 5] (order,type,nq,nme) : (1,orthotope,1,1)
    [ 6] (order,type,nq,nme) : (1,orthotope,1,1)
    [ 7] (order,type,nq,nme) : (1,orthotope,1,1)
  ```

  .
- `order = value` : gives the order of the mesh elements (default is 1).

Listing 3: : OrthMesh constructor in dimension d=3 (simplicial mesh)

```
Oh = fc_hypermesh.OrthMesh(2,[10,20],order=4)
print(Oh)
```

Output

```
OrthMesh object
      d : 2
   order : 4
     box : [[0.0, 1.0], [0.0, 1.0]]
 mapping : None
Mesh (order,type,nq,nme) : (4,simplex,3321,400)
Number of 1-faces : 4
  [ 0] (order,type,nq,nme) : (4,simplex,81,20)
  [ 1] (order,type,nq,nme) : (4,simplex,81,20)
  [ 2] (order,type,nq,nme) : (4,simplex,41,10)
  [ 3] (order,type,nq,nme) : (4,simplex,41,10)
Number of 0-faces : 4
  [ 0] (order,type,nq,nme) : (1,simplex,1,1)
  [ 1] (order,type,nq,nme) : (1,simplex,1,1)
  [ 2] (order,type,nq,nme) : (1,simplex,1,1)
  [ 3] (order,type,nq,nme) : (1,simplex,1,1)
```

- `box = value` : used to specify the d-orthotope $[a_1, b_1] \times \ldots \times [a_{\mathrm{d}}, b_{\mathrm{d}}]$ by setting `value` as an d-by-2 array such that $a_i =$ `value(i,1)` and $b_i =$ `value(i,2)`.

Listing 4: : OrthMesh constructor in dimension d=3 (simplicial mesh)

```
Oh = fc_hypermesh.OrthMesh(2,10, box=[[-1,1],[-2,2]])
print('Oh␣=\n'+str(Oh))
```

Output

```
Oh =
OrthMesh object
      d : 2
   order : 1
     box : [[-1.0, 1.0], [-2.0, 2.0]]
 mapping : None
Mesh (order,type,nq,nme) : (1,simplex,121,200)
Number of 1-faces : 4
  [ 0] (order,type,nq,nme) : (1,simplex,11,10)
  [ 1] (order,type,nq,nme) : (1,simplex,11,10)
  [ 2] (order,type,nq,nme) : (1,simplex,11,10)
  [ 3] (order,type,nq,nme) : (1,simplex,11,10)
Number of 0-faces : 4
  [ 0] (order,type,nq,nme) : (1,simplex,1,1)
  [ 1] (order,type,nq,nme) : (1,simplex,1,1)
  [ 2] (order,type,nq,nme) : (1,simplex,1,1)
  [ 3] (order,type,nq,nme) : (1,simplex,1,1)
```

- `m_min = value` : used to only build the `m`-Faces for `m` in $[\![$`m_min`$, \mathrm{d}[\![$. Default `value` is 0.

Listing 5: : OrthMesh constructor in dimension d=3 (simplicial mesh)

```
Oh = fc_hypermesh.OrthMesh(3,10, m_min=2)
print('Oh␣=\n'+str(Oh))
print('q␣=\n'+str(Oh.Mesh.q))
```

Output

```
Oh =
OrthMesh object
      d : 3
   order : 1
     box : [[0.0, 1.0], [0.0, 1.0], [0.0, 1.0]]
 mapping : None
Mesh (order,type,nq,nme) : (1,simplex,1331,6000)
Number of 2-faces : 6
  [ 0] (order,type,nq,nme) : (1,simplex,121,200)
  [ 1] (order,type,nq,nme) : (1,simplex,121,200)
  [ 2] (order,type,nq,nme) : (1,simplex,121,200)
  [ 3] (order,type,nq,nme) : (1,simplex,121,200)
  [ 4] (order,type,nq,nme) : (1,simplex,121,200)
  [ 5] (order,type,nq,nme) : (1,simplex,121,200)

q =
[[0.  0.1 0.2 ... 0.8 0.9 1. ]
 [0.  0.  0.  ... 1.  1.  1. ]
 [0.  0.  0.  ... 1.  1.  1. ]]
```

- `mapping=value` : used to apply on the mesh a mapping function given by a function handle.

Listing 6: : OrthMesh constructor in dimension d=3 (simplicial mesh)

```
import numpy as np
mfun = lambda q: np.array([q[0]+np.sin(q[1]),q[1],q[2]])
Oh = fc_hypermesh.OrthMesh(3,10, m_min=2, mapping=mfun)
print('Oh␣=\n'+str(Oh))
print('q␣=\n'+str(Oh.Mesh.q))
```

Output

```
Oh =
 OrthMesh object
        d : 3
    order : 1
      box : [[0.0, 1.8414709848078965], [0.0, 1.0], [0.0, 1.0]]
  mapping : lambda q: np.array([q[0]+np.sin(q[1]),q[1],q[2]])
 Mesh (order,type,nq,nme) : (1,simplex,1331,6000)
 Number of 2-faces : 6
   [ 0] (order,type,nq,nme) : (1,simplex,121,200)
   [ 1] (order,type,nq,nme) : (1,simplex,121,200)
   [ 2] (order,type,nq,nme) : (1,simplex,121,200)
   [ 3] (order,type,nq,nme) : (1,simplex,121,200)
   [ 4] (order,type,nq,nme) : (1,simplex,121,200)
   [ 5] (order,type,nq,nme) : (1,simplex,121,200)

q =
[[0.        0.1        0.2        ... 1.64147098 1.74147098 1.84147098]
 [0.        0.         0.         ... 1.         1.         1.        ]
 [0.        0.         0.         ... 1.         1.         1.        ]]
```

3.2.2 Access to OrthMesh's fields

In all examples given in this section, Oh is the OrthMesh object given by

```
Oh = fc_hypermesh.OrthMesh(3,10, order=2)
print(Oh)
```

Output

```
OrthMesh object
       d : 3
   order : 2
     box : [[0.0, 1.0], [0.0, 1.0], [0.0, 1.0]]
 mapping : None
Mesh (order,type,nq,nme) : (2,simplex,9261,6000)
Number of 2-faces : 6
  [ 0] (order,type,nq,nme) : (2,simplex,441,200)
  [ 1] (order,type,nq,nme) : (2,simplex,441,200)
  [ 2] (order,type,nq,nme) : (2,simplex,441,200)
  [ 3] (order,type,nq,nme) : (2,simplex,441,200)
  [ 4] (order,type,nq,nme) : (2,simplex,441,200)
  [ 5] (order,type,nq,nme) : (2,simplex,441,200)
Number of 1-faces : 12
  [ 0] (order,type,nq,nme) : (2,simplex,21,10)
  [ 1] (order,type,nq,nme) : (2,simplex,21,10)
  [ 2] (order,type,nq,nme) : (2,simplex,21,10)
  [ 3] (order,type,nq,nme) : (2,simplex,21,10)
  [ 4] (order,type,nq,nme) : (2,simplex,21,10)
  [ 5] (order,type,nq,nme) : (2,simplex,21,10)
  [ 6] (order,type,nq,nme) : (2,simplex,21,10)
  [ 7] (order,type,nq,nme) : (2,simplex,21,10)
  [ 8] (order,type,nq,nme) : (2,simplex,21,10)
  [ 9] (order,type,nq,nme) : (2,simplex,21,10)
  [10] (order,type,nq,nme) : (2,simplex,21,10)
  [11] (order,type,nq,nme) : (2,simplex,21,10)
Number of 0-faces : 8
  [ 0] (order,type,nq,nme) : (1,simplex,1,1)
  [ 1] (order,type,nq,nme) : (1,simplex,1,1)
  [ 2] (order,type,nq,nme) : (1,simplex,1,1)
  [ 3] (order,type,nq,nme) : (1,simplex,1,1)
  [ 4] (order,type,nq,nme) : (1,simplex,1,1)
  [ 5] (order,type,nq,nme) : (1,simplex,1,1)
  [ 6] (order,type,nq,nme) : (1,simplex,1,1)
  [ 7] (order,type,nq,nme) : (1,simplex,1,1)
```

It's a 3 dimensional mesh of the unit cube tesselated with 2-order simplices where their vertices are in $(i/10, j/10, k/10)$ for all $(i, j, k) \in [\![0, 10]\!]$. The main mesh given as an EltMesh object is Oh.Mesh

```
Th=Oh.Mesh
print('Th␣=␣'+ str(Th))
```

Output

```
Th = EltMesh object
    type : 0 (simplex)
   order : 2
   label : 1
       d : 3
       m : 3
       q : (3,9261)
      me : (10,6000)
```

The k^{th} m-faces of `Oh` stored as an `EltMesh` object is given by `Oh.Faces[Oh.d-m-1][k-1]`.

```
Fh=Oh.Faces[0][2]
print('Fh = '+ str(Fh))
```

Output

```
Fh = EltMesh object
   type : 0 (simplex)
  order : 2
  label : 3
      d : 3
      m : 2
      q : (3,441)
     me : (6,200)
```

One can easily access to each field of an `EltMesh` object (see section 3.1). For example, to acces the nodes array and the connectivity array of the `Th` `EltMesh` object we do respectively `Th.q` and `Th.me`. We also have the following link between global nodes array `Th.q` and local nodes array `Fh.q`:

`Th.q[:,Fh.toGlobal]==Fh.q`

or more generaly, for all `m` in $[\![0, \texttt{Oh.d}]\!]$ and for all `k` in $[\![1, E_m^{\mathrm{d}}]\!]$

`Oh.Mesh.q[:,Oh.Faces[Oh.d-m-1][k-1].toGlobal]==Oh.Faces[Oh.d-m-1][k-1].q`

where E_m^{d} is defined in (1).

3.2.3 `plotmesh` method

The `plotmesh()` member function uses Matplotlib Python package [2, 3] to represent the mesh given by an `OrthMesh` object.

Syntaxe

```
obj.plotmesh()
obj.plotmesh(key=value, ...)
```

Description

`obj.plotmesh()` plot the main mesh.

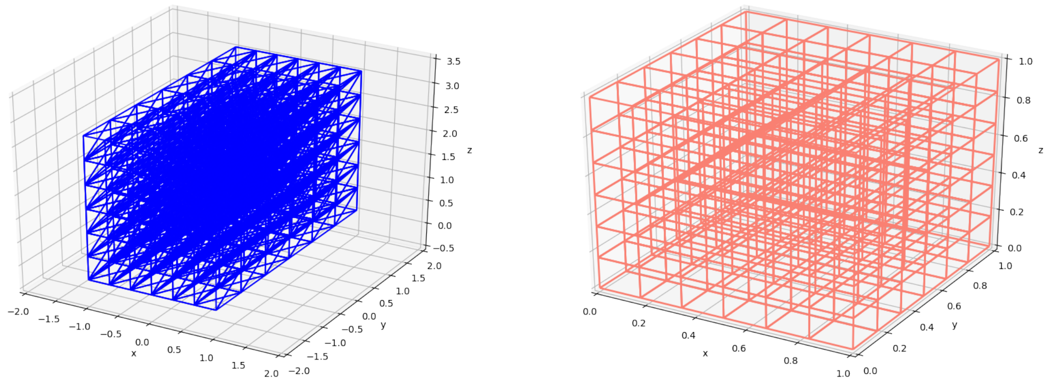
`obj.plotmesh(key=value, ...)`
Some optional `key`/`value` pairs arguments are available with `key`:

- `legend` : if `value` is `True`, a legend is displayed. Default is `False`.
- `m` : plots all the `m`-faces of the mesh. Default `m = d` i.e. the main mesh. ($0 \leqslant$`m`$\leqslant$`d`)
- `labels` : plot all the `m`-faces of the mesh with number/label in `value` list
- `color` : use to specify the color to use.
- ...

Other `key`/`value` pairs arguments can be used depending of `obj.d` and `obj.m` values and they are those of the Matplotlib function used:

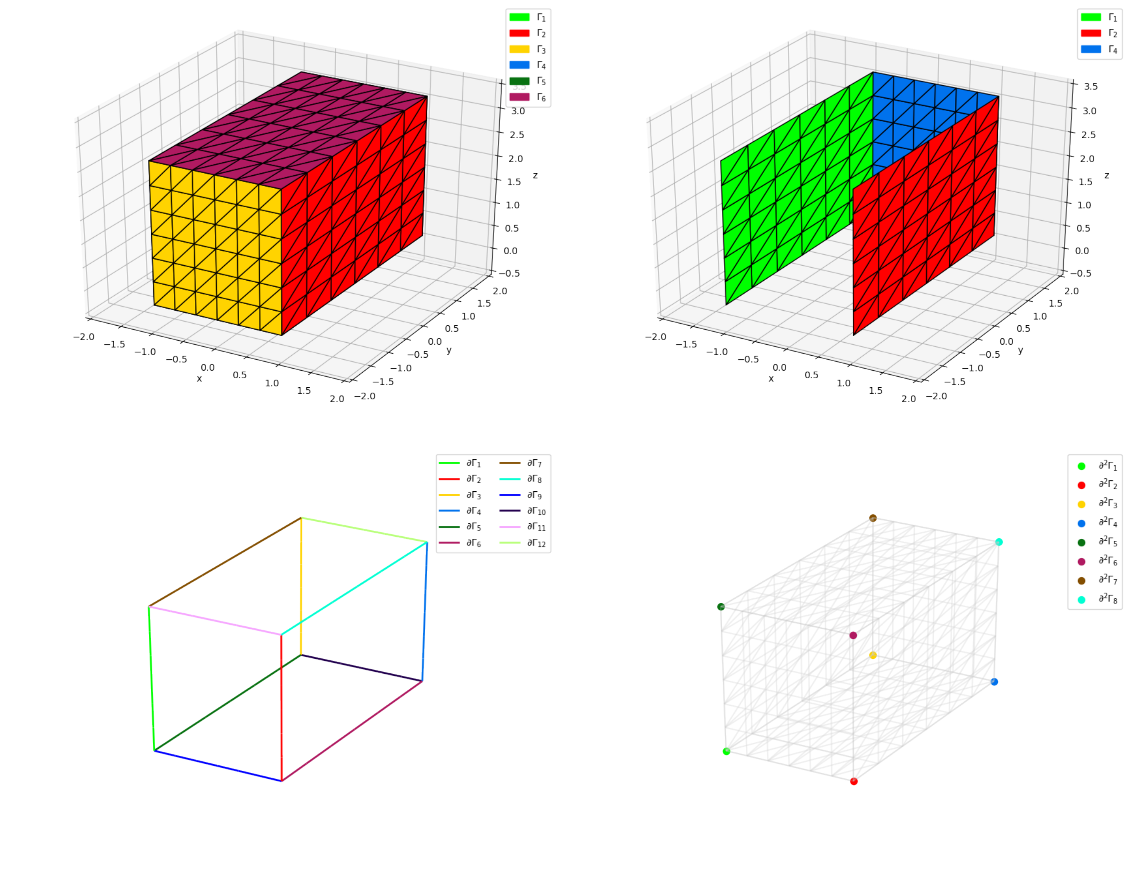
- with `obj.d=3` and `obj.m=3`, `Line3DCollection` function is used;
- with `obj.d=3` and `obj.m=2`, `Poly3DCollection` or `Line3DCollection` function are used;
- with `obj.d=3` and `obj.m=1`, `Line3DCollection` function is used;
- with `obj.d=3` and `obj.m=0`, `scatter` function is used;
- with `obj.d=2` and `obj.m=2`, `PolyCollection` function is used;
- with `obj.d=2` and `obj.m=1`, `plot` function is used;
- with `obj.d=2` and `obj.m=0`, `scatter` function is used;
- with `obj.d=1` and `obj.m=1`, `plot` function is used;
- with `obj.d=1` and `obj.m=0`, `scatter` function is used;

Examples In Listing 7 and Listing 8, the code given in Listing 1 is supposed to be preloaded.



```
Oh1=fc_hypermesh.OrthMesh(3,6,box=[[-1,1],[-2,2],[0,3]])
plt.figure(1)
Oh1.plotmesh()
set_axes_equal()
Oh2=fc_hypermesh.OrthMesh(3,6,type='orthotope')
plt.figure(2)
Oh2.plotmesh(color='Salmon', linewidth=2)
set_axes_equal()
```

Listing 7: `plotmesh` method of 3D `OrthMesh` objects tessellated with simplices in `figure(1)` (left) and with orthotopes in `figure(2)` (right)




```
Oh=fc_hypermesh.OrthMesh(3,6,box=[[-1,1],[-2,2],[0,3]]);
plt.figure(1)
Oh.plotmesh(m=2,legend=True)
set_axes_equal()
plt.figure(2)
Oh.plotmesh(m=2,legend=True,labels=[1,2,4])
set_axes_equal()
plt.figure(3)
Oh.plotmesh(m=1,legend=True,linewidth=2)
set_axes_equal();plt.axis('off')
plt.figure(4)
Oh.plotmesh(m=2,facecolor=None,edgeColor='LightGrey',alpha=0.2)
Oh.plotmesh(m=1,color='DarkGrey',alpha=0.2)
Oh.plotmesh(m=0,legend=True,s=50)
set_axes_equal();plt.axis('off')
```

Listing 8: `plotmesh` method of a 3D `OrthMesh` object, `figure(1)` (upper left), `figure(2)` (upper right),`figure(3)` (bottom left), `figure(4)` (bottom right)

3.2.4 `plotnodes` method

The `plotnodes()` member function can be used to represent nodes of the mesh given by an `OrthMesh` object if the space dimension d is less than or equal to 3.

Syntaxe

```
obj.plotnodes()
obj.plotnodes(key=value, ...)
```

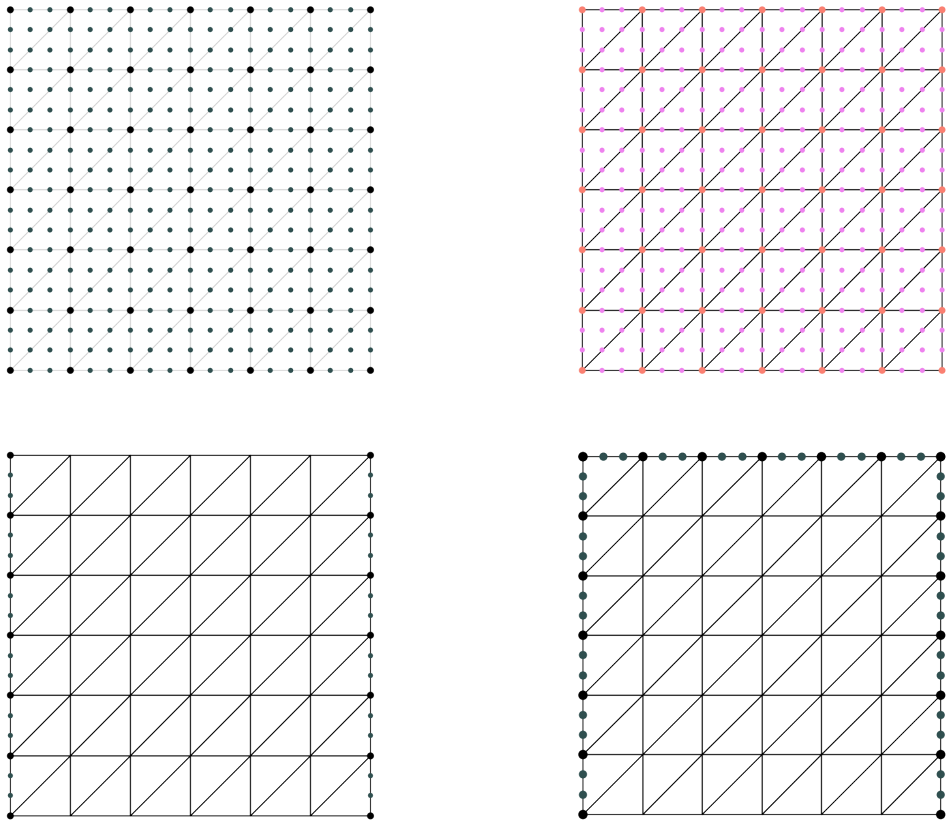
Description

`obj.plotnodes()`
Uses Matplotlib `scatter` function to represent nodes of the mesh as points. Vertices of the mesh elements are also nodes and they are distinguishable from others nodes.

`obj.plotnodes(key=value, ...)`
Some optional `key`/`value` pairs arguments are available with `key`:

- `m` : plots all the nodes of the `m`-faces of the mesh. Default `m = d` i.e. the main mesh. ($0 \leqslant$ `m` $\leqslant d$)
- `labels` : plot all the nodes of the `m`-faces of the mesh with number/label in `value` list
- `vcolor` : use to specify the point color for the mesh vertices. Default is `obj.color`.
- `vsize` : use to specify the point size for the mesh vertices. Default is `40`.
- `ncolor` : use to specify the color of the nodes (not vertices) of the mesh elements. Default is `'k'` (ie. black).
- `nsize` : use to specify the size of the nodes (not vertices) of the mesh elements. Default is `30`.

Other `key`/`value` pairs arguments can be used: they are those of the `scatter` function.



```
Oh=fc_hypermesh.OrthMesh(2,6,order=3)
plt.figure(1)
Oh.plotmesh(color='LightGray')
Oh.plotnodes()
set_axes_equal();plt.axis('off')
plt.figure(2)
Oh.plotmesh(color='k')
Oh.plotnodes(vcolor='Salmon',ncolor='Violet')
set_axes_equal();plt.axis('off')
plt.figure(3)
Oh.plotmesh(color='k')
Oh.plotnodes(m=1,labels=[1,2])
set_axes_equal();plt.axis('off')
plt.figure(4)
Oh.plotmesh(color='k')
Oh.plotnodes(m=1,labels=[1,2,4],vsize=80,nsize=60)
set_axes_equal();plt.axis('off')
```

Listing 9: `plotnodes` method of a 2D `OrthMesh` object, `figure(1)` (upper left), `figure(2)` (upper right), `figure(3)` (bottom left), `figure(4)` (bottom right)

3.2.5 `ploteltsNumber` method

The `ploteltsNumber()` member function can be used to display elements index/number of the mesh given by an `OrthMesh` object if the space dimension d is less than or equal to 3.

Syntaxe

```
obj.plotnodesNumber()
obj.plotnodesNumber(key=value, ...)
```

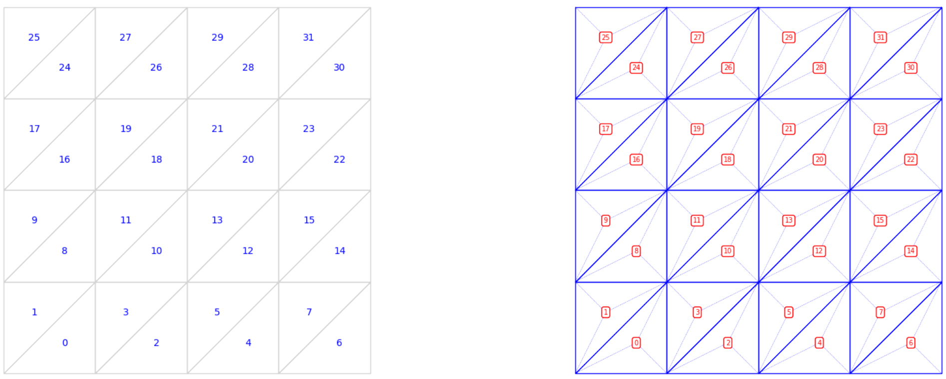
Description

`obj.plotnodesNumber()`
: Uses `fc_hypermesh.ploteElementsNumber` function to represent node numbers.

`obj.ploteltsNumber(key, value, ...)`
: Some optional `key`/`value` pairs arguments are available with `key`:

- `m` : plots all the nodes index/number of the `m`-faces of the mesh. Default `m = d` i.e. the main mesh. $(0 \leqslant$ `m` $\leqslant d)$
- `labels` : plot all the elements index/number of the `m`-faces of the mesh with number/label in `value` list.
- `color` : use to specify text color. Default is `'auto'` to automaticaly set to the color of the main mesh (if `m = d`) or to the colors of the `m`-faces currently drawn.
- `ec` : use to specify the color of box outline. Default is `'w'` (i.e. white). Value `'auto'` can be used.
- `fc` : use to specify text background color. Default is `'w'` (i.e. white). Value `'auto'` can be used.
- `vLineColor` : Draw lines with `value` as color beetween vertices and barycenter of the mesh elements. Default is `None` (no lines).
- `vLineStyle` : Select lines type. Default is `':'` (dotted lines).
- `vLineWidth` : Set lines witdh. Default is `0.5`.

Other `key`/`value` pairs arguments can be used: they are those of the Matplotlib `text` function in 3D or those of the Matplotlib `annotate` function in 2D.



```
Oh=fc_hypermesh.OrthMesh(2,4,order=3)
plt.figure(1)
Oh.plotmesh(color='LightGray')
Oh.ploteltsNumber()
set_axes_equal();plt.axis('off')
plt.figure(2)
Oh.plotmesh()
Oh.ploteltsNumber(color='r',ec='r', fontsize=7, vLineColor='auto', fc='w')
set_axes_equal();plt.axis('off')
```

Listing 10: `ploteltsNumber` method of a 2D `EltMesh` objects , `figure(1)` (left) and `figure(2)` (right)

3.2.6 `ploteltsNumber` method

The `ploteltsNumber()` member function can be used to display elements index/number of the mesh given by an `OrthMesh` object if the space dimension d is less than or equal to 3.

Syntaxe

```
obj.ploteltsNumber()
obj.ploteltsNumber(key=value, ...)
```

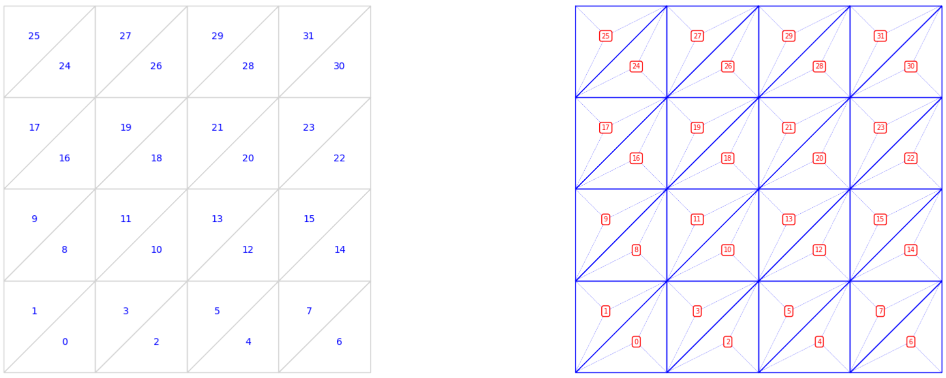
Description

`obj.ploteltsNumber()`
Uses `fc_hypermesh.ploteElementsNumber` function to represent node numbers.

`obj.ploteltsNumber(key, value, ...)`
Some optional `key`/`value` pairs arguments are available with `key`:

- `m` : plots all the elements index/number of the `m`-faces of the mesh. Default `m = d` i.e. the main mesh. $(0 \leqslant$ `m` $\leqslant \mathrm{d})$
- `labels` : plot all the elements index/number of the `m`-faces of the mesh with number/label in `value` list.
- `color` : use to specify text color. Default is `'auto'` to automaticaly set to the color of the main mesh (if `m = d`) or to the colors of the `m`-faces currently drawn.
- `ec` : use to specify the color of box outline. Default is `'w'` (i.e. white). Value `'auto'` can be used.
- `fc` : use to specify text background color. Default is `'w'` (i.e. white). Value `'auto'` can be used.
- `vLineColor` : Draw lines with `value` as color beetween vertices and barycenter of the mesh elements. Default is `None` (no lines).
- `vLineStyle` : Select lines type. Default is `':'` (dotted lines).
- `vLineWidth` : Set lines witdh. Default is `0.5`.

Other `key`/`value` pairs arguments can be used: they are those of the Matplotlib `text` function in 3D or those of the Matplotlib `annotate` function in 2D.



```
Oh=fc_hypermesh.OrthMesh(2,4,order=3)
plt.figure(1)
Oh.plotmesh(color='LightGray')
Oh.ploteltsNumber()
set_axes_equal();plt.axis('off')
plt.figure(2)
Oh.plotmesh()
Oh.ploteltsNumber(color='r',ec='r', fontsize=7, vLineColor='auto', fc='w')
set_axes_equal();plt.axis('off')
```

Listing 11: `ploteltsNumber` method of a 2D `EltMesh` objects , `figure(1)` (left) and `figure(2)` (right)

4 Using the fc hypermesh package

Before using this class it will be necessary to be aware of the memory used by this one. For example, when meshing a 6-dimensional orthotope with 1-order simplices by taking `N` = 10 intervals in each space direction, gives an `OrthMesh` object using `48.096 GB` in memory. With 3-order simplices, the `OrthMesh` object uses `617.764 GB` in memory!

The memory usage for a d-dimensional `OrthMesh` object by taking `N` = 10 intervals in each space direction is given in Table 3 for 1-order elements and in Table 4 for 3-order elements. One can refer to Section 5 for more details.

d	`OrthMesh` (orthotopes)	`OrthMesh` (simplices)
1	296 B	296 B
2	6 KB	8 KB
3	144 KB	282 KB
4	2 MB	12 MB
5	57 MB	696 MB
6	1.163 GB	48.096 GB
7	23.815 GB	3.845 TB
8	496.025 GB	346.756 TB

Table 3: Memory usage of `OrthMesh` object for the tessellation of an orthotope by 1-order orthotopes and by 1-order simplices according to the space dimension d and with $N = 10$.

d	`OrthMesh` (orthotopes)	`OrthMesh` (simplices)
1	616 B	616 B
2	32 KB	35 KB
3	1 MB	1 MB
4	64 MB	115 MB
5	2.695 GB	7.737 GB
6	109.134 GB	617.764 GB
7	4.352 TB	58.136 TB
8	171.900 TB	6247.498 TB

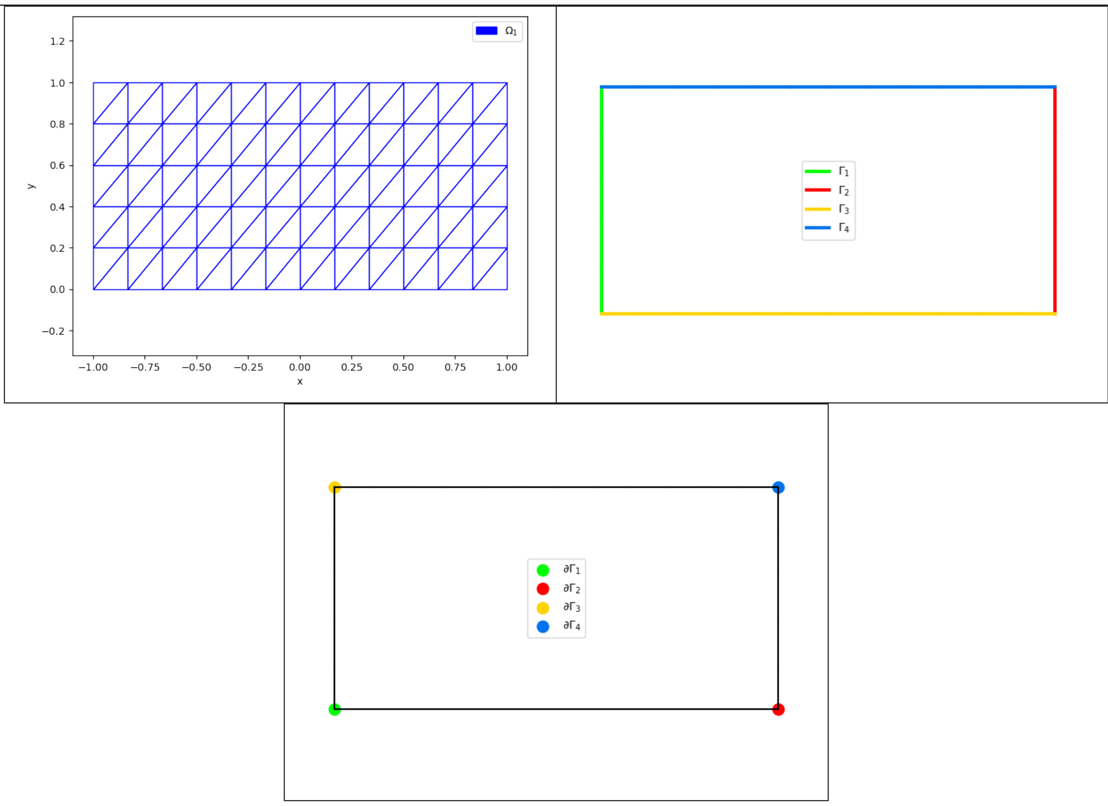
Table 4: Memory usage of `OrthMesh` object for the tessellation of an orthotope by 3-order orthotopes and by 3-order simplices according to the space dimension d and with $N = 10$.

In all the next examples, the following code is previously load:

```
import matplotlib.pyplot as plt
from fc_tools.colors import str2rgb
from fc_hypermesh import OrthMesh
from fc_tools.Matplotlib import DisplayFigures,set_axes_equal
```

4.1 2d-orthotope meshing by simplices

In Listing 12, an `OrthMesh` object is built under Python for the orthotope $[-1, 1] \times [0, 1]$ with simplicial elements and $\mathbf{N} = (12, 5)$. The main mesh and all the m-face meshes of the resulting object are plotted.

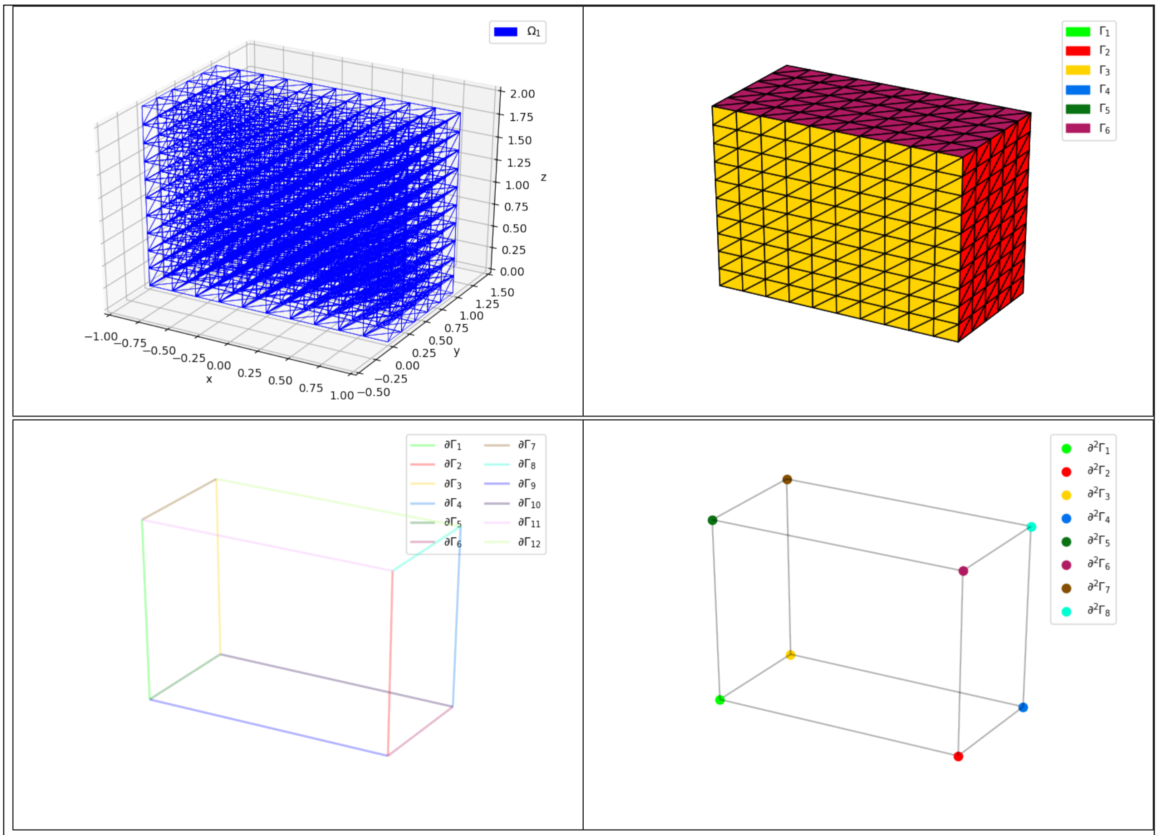



```
oTh=OrthMesh(2,[12,5],type='simplex',box=[[-1,1],[0,1]])
plt.figure(1)
oTh.plotmesh(legend=True)
set_axes_equal()
plt.figure(2)
oTh.plotmesh(m=1,legend=True,linewidth=3)
plt.axis('off')
set_axes_equal()
plt.figure(3)
oTh.plotmesh(m=1,color='black')
oTh.plotmesh(m=0,legend=True,s=105)
plt.axis('off')
set_axes_equal()
```

Listing 12: 2D simplicial `OrthMesh` object with Python 3.8.1, main mesh (figure 1, upper left), 1-face meshes (figure 2, upper right), and 0-face meshes (figure 3, bottom)

4.2 3d-orthotope meshing by simplices

In Listing 13, an `OrthMesh` object is built under Python for the orthotope $[-1,1] \times [0,1] \times [0,2]$ with simplicial elements and $\mathbf{N} = (10, 5, 10)$. The main mesh and all the m-face meshes of the resulting object are plotted.

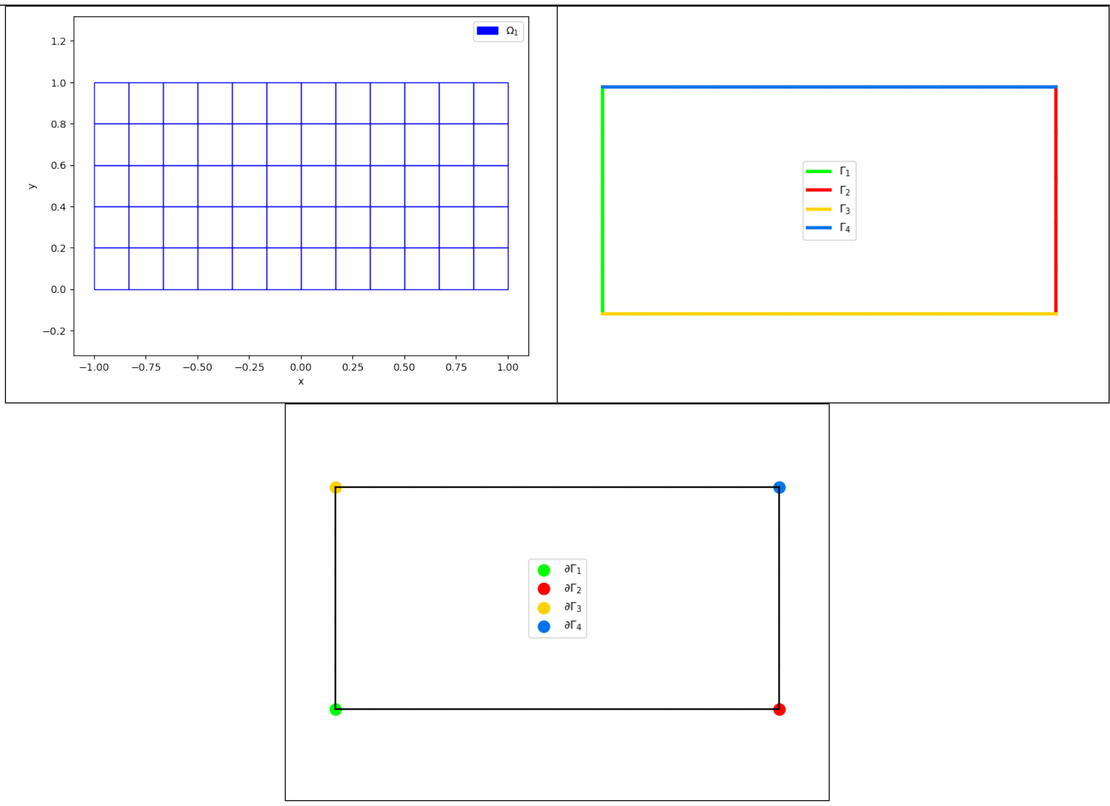



```
oTh=OrthMesh(3,[10,5,10],box=[[-1,1],[0,1],[0,2]])
plt.figure(1)
oTh.plotmesh(legend=True,linewidth=0.5)
set_axes_equal()
plt.figure(2)
oTh.plotmesh(m=2,legend=True,edgecolor=[0,0,0])
plt.axis('off')
set_axes_equal()
plt.figure(3)
oTh.plotmesh(m=2,edgecolor=[0,0,0],color='none')
oTh.plotmesh(m=1,legend=True,linewidth=2,alpha=0.3)
plt.axis('off')
set_axes_equal()
plt.figure(4)
oTh.plotmesh(m=1,color='black',alpha=0.3)
oTh.plotmesh(m=0,legend=True,s=55)
set_axes_equal()
plt.axis('off')
```

Listing 13: 3D simplicial `OrthMesh` object with Python 3.8.1, main mesh (figure 1, upper left), 2-face meshes (figure 2, upper right), 1-face meshes (figure 3, bottom left) and 0-face meshes (figure 4, bottom right)

4.3 2d-orthotope meshing by orthotopes

In Listing 14, an `OrthMesh` object is built under Python for the orthotope $[-1,1]\times[0,1]$ with orthotope elements and $\mathbf{N}=(10,5,10)$. The main mesh and all the m-face meshes of the resulting object are plotted.

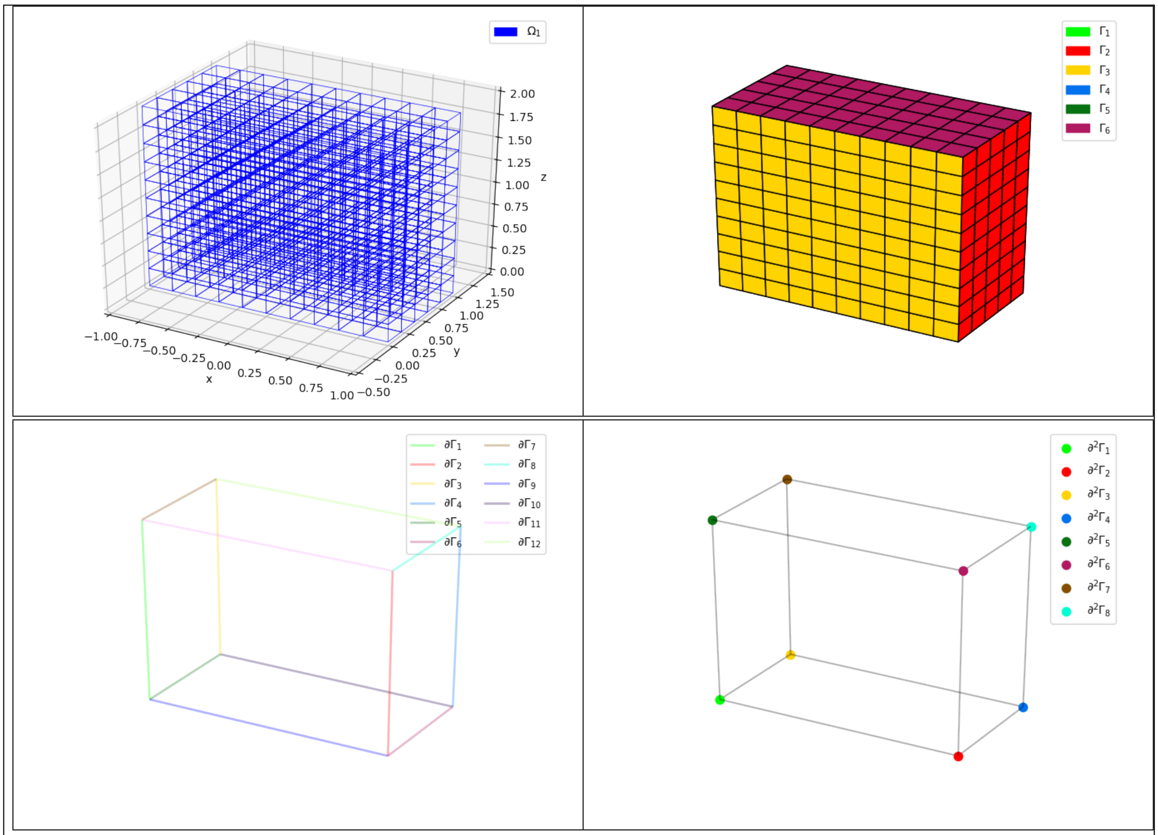



```
oTh=OrthMesh(2,[12,5],type='orthotope',box=[[-1,1],[0,1]])
plt.figure(1)
oTh.plotmesh(legend=True)
set_axes_equal()
plt.figure(2)
oTh.plotmesh(m=1,legend=True,linewidth=3)
plt.axis('off')
set_axes_equal()
plt.figure(3)
oTh.plotmesh(m=1,color='black')
oTh.plotmesh(m=0,legend=True,s=105)
plt.axis('off')
set_axes_equal()
```

Listing 14: 2D orthotope `OrthMesh` object with Python 3.8.1, main mesh (upper left), 1-face meshes (upper right), and 0-face meshes (bottom)

4.4 3d-orthotope meshing by orthotopes

In Listing 15, an `OrthMesh` object is built under Python for the orthotope $[-1,1] \times [0,1] \times [0,2]$ with orthotope elements and $\mathbf{N} = (10,5,10)$. The main mesh and all the m-face meshes of the resulting object are plotted.



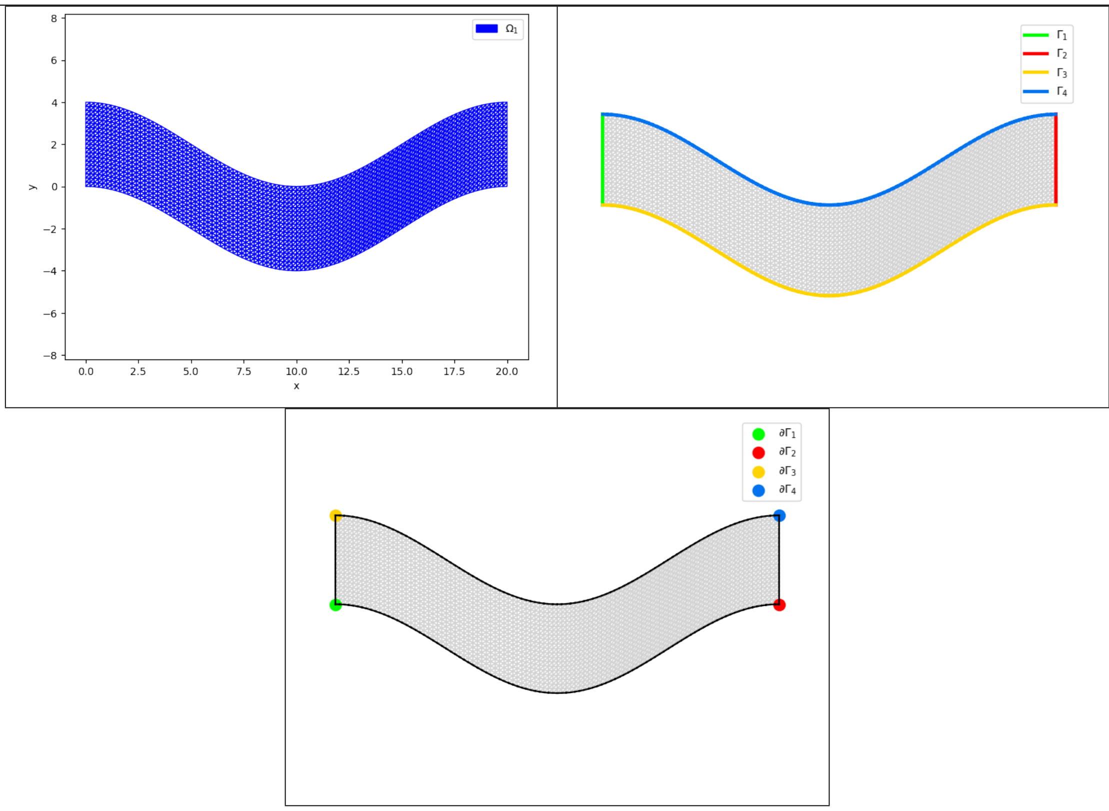

```
oTh=OrthMesh(3,[10,5,10],type='orthotope', box=[[-1,1],[0,1],[0,2]])
plt.figure(1)
oTh.plotmesh(legend=True,linewidth=0.5)
set_axes_equal()
plt.figure(2)
oTh.plotmesh(m=2,legend=True,edgecolor=[0,0,0])
plt.axis('off')
set_axes_equal()
plt.figure(3)
oTh.plotmesh(m=2,edgecolor=[0,0,0],color='none')
oTh.plotmesh(m=1,legend=True,linewidth=2,alpha=0.3)
plt.axis('off')
set_axes_equal()
plt.figure(4)
oTh.plotmesh(m=1,color='black',alpha=0.3)
oTh.plotmesh(m=0,legend=True,s=55)
set_axes_equal()
plt.axis('off')
```

Listing 15: 3D orthotope `OrthMesh` object with Python 3.8.1, main mesh (upper left), 2-face meshes (upper right), 1-face meshes (bottom left) and 0-face meshes (bottom right)

4.5 Mapping of a 2d-orthotope meshing by simplices

For example, the following 2D geometrical transformation allows to deform the reference unit hypercube.

$$\begin{array}{ccc} [0,1]\times[0,1] & \longrightarrow & \mathbb{R}^2 \\ \begin{pmatrix} x \\ y \end{pmatrix} & \longrightarrow & F(x,y) = \begin{pmatrix} 20\,x \\ 2\,(2\,y - 1 + \cos(2\pi x)) \end{pmatrix} \end{array}$$



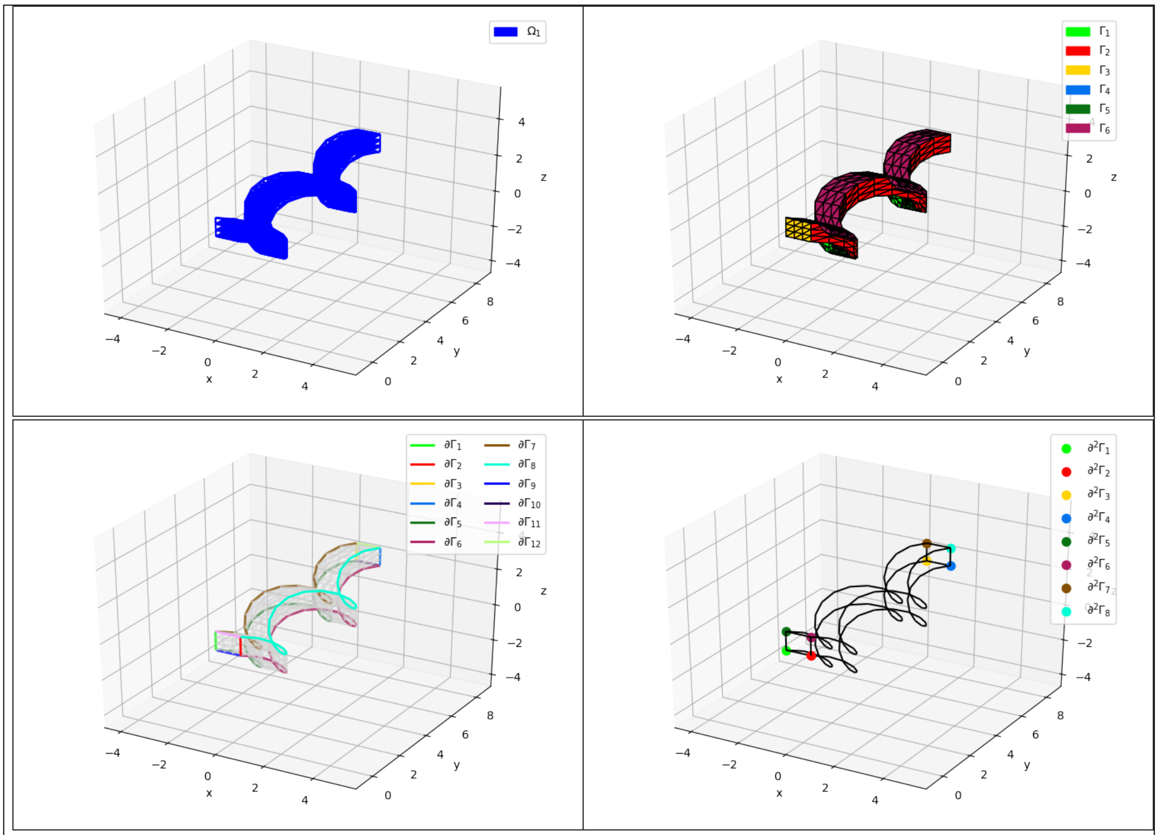

```
import numpy as np
trans=lambda q: np.array([20*q[0],2*(2*q[1]-1+np.cos(2*np.pi*q[0]))])
oTh=OrthMesh(2,[100,20],type='simplex',mapping=trans)
plt.figure(1)
oTh.plotmesh(legend=True)
plt.axis('equal')
plt.figure(2)
oTh.plotmesh(color='lightgray')
oTh.plotmesh(m=1,legend=True,linewidth=3)
plt.axis('equal')
plt.axis('off')
plt.figure(3)
oTh.plotmesh(color='lightgray')
oTh.plotmesh(m=1,color='black')
oTh.plotmesh(m=0,legend=True,s=105)
plt.axis('equal')
plt.axis('off')
```

Listing 16: Mapping of a 2D simplicial `OrthMesh` object with Python 3.8.1, main mesh (upper left), 1-face meshes (upper right), and 0-face meshes (bottom)

4.6 Mapping of a 3d-orthotope meshing by orthotopes

For example, the following 3D geometrical transformation allows to deform the reference unit hypercube.

$$\begin{array}{ccc} [0,1]\times[0,1]\times[0,1] & \longrightarrow & \mathbb{R}^2 \\ \begin{pmatrix} x \\ y \\ z \end{pmatrix} & \longrightarrow & F(x,y,y) = \begin{pmatrix} x+\sin(4\pi y) \\ 10y \\ z+\cos(4\pi y) \end{pmatrix} \end{array}$$




```
import numpy as np
trans=lambda q: np.array([q[0]+np.sin(4*np.pi*q[1]), 10*q[1]-1,
    q[2]+np.cos(4*np.pi*q[1])])
oTh=OrthMesh(3,[3,25,3],type='simplex',mapping=trans)
plt.figure(1)
oTh.plotmesh(legend=True)
set_axes_equal()
plt.figure(2)
oTh.plotmesh(m=2,legend=True,edgecolor=[0,0,0])
set_axes_equal()
plt.figure(3)
oTh.plotmesh(m=2,edgecolor='lightgray',facecolor=None,alpha=0.3)
oTh.plotmesh(m=1,legend=True,linewidth=2)
set_axes_equal()
plt.figure(4)
oTh.plotmesh(m=1,color='black')
oTh.plotmesh(m=0,legend=True,s=55)
set_axes_equal()
```

Listing 17: Mapping of a 3D orthotope `OrthMesh` object with Python 3.8.1, main mesh (upper left), 2-face meshes (upper right), 1-face meshes (bottom left) and 0-face meshes (bottom right)

5 Memory consuming

Take care when using theses codes with memory consuming : the number of points $\mathrm{n_q}$ and the number of elements increases exponentially according to the space dimension d. If $(N+1)$ points are taken in each space direction, we have

$$\mathrm{n_q} = (\mathtt{p}N+1)^d, \quad \text{for both tessellation and triangulation}$$

and

$$\begin{aligned} \mathrm{n_{me}} &= N^d, && \text{for tessellation by orthotopes} \\ \mathrm{n_{me}} &= d!N^d, && \text{for tessellation by simplices.} \end{aligned}$$

If the array q is stored as *double* (8 octets) then

$$\text{mem. size of q} = \text{d} \times \text{n}_\text{q} \times 8 \text{ octets}$$

and if the array me as *int* (8 octets) then

$$\text{mem. size of me} = \begin{cases} 2^\text{d} \times \text{n}_\text{me} \times 8 \text{ octets} & \text{(tessellation by orthotopes)} \\ (\text{d}+1) \times \text{n}_\text{me} \times 8 \text{ octets} & \text{(tessellation by simplices)} \end{cases}$$

For $N = 10$ and $\text{d} \in [\![1, 8]\!]$, the values of n_q and n_me are given in Table 5. The memory usage for the corresponding array q and array me is available in Table 6.

d	$\text{n}_\text{q} = (N+1)^\text{d}$	$\text{n}_\text{me} = N^\text{d}$ (orthotopes)	$\text{n}_\text{me} = \text{d}!N^\text{d}$ (simplices)
1	11	10	10
2	11	100	200
3	11	1 000	6 000
4	11	10 000	240 000
5	11	100 000	12 000 000
6	11	1 000 000	720 000 000
7	11	10 000 000	50 400 000 000
8	11	100 000 000	4 032 000 000 000

Table 5: Number of vertices n_q and number of elements n_me for the tessellation of an orthotope by orthotopes and by simplices according to the space dimension d and with $N = 10$.

d	q	me (orthotopes)	me (simplices)
1	88 o	80 o	80 o
2	176 o	1 ko	2 ko
3	264 o	32 ko	96 ko
4	352 o	640 ko	4 Mo
5	440 o	12 Mo	288 Mo
6	528 o	256 Mo	20 Go
7	616 o	5 Go	1 To
8	704 o	102 Go	145 To

Table 6: Memory usage of the array q and the array me for the tessellation of an orthotope by orthotopes and by simplices according to the space dimension d and with $N = 10$.

6 Benchmarks

For all the following tables, the computational costs of the `OrthMesh` constructor are given for the orthotope $[-1, 1]^\text{d}$ under Python 3.8.1. The computations were done on a laptop with `Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz` processor and 63Go of RAM under `Ubuntu 18.04.3 LTS` (x86_64).

In the following pages, computational costs of the `OrthMesh` constructor will be done by using `bench01` function. As sample, we give an example with output. Thereafter, all the output will be presented in tabular form.

```
from fc_hypermesh import  bench
bench(3,range(20,170,20),type='simplex',box=[[-1,1],[-1,1],[-1,1]],order=1)
```

Listing 18: bench sample

Output

```
#------------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.3 LTS (x86_64)
#   processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#              (1 procs/14 cores by proc/2 threads by core)
#         RAM: 62.6 Go
#    software: Python
#     release: 3.8.1
#------------------------------------------------------------------------------
# fc_hypermesh.OrthMesh constructor with
#   d      =3
#   type   =simplex
#   order  =1
#   box    =[[-1, 1], [-1, 1], [-1, 1]]
#   mapping=None
#------------------------------------------------------------------------------
#date:2019-12-30_09-33-14
#nbruns:5
#numpy:      i8         i8         i8            f8
#format: {:>7d}   {:>10d}   {:>10d}      {:11.3f}
#labels:       N         nq        nme   OrthMesh(s)
              20       9261      48000         0.185
              40      68921     384000         0.190
              60     226981    1296000         0.268
              80     531441    3072000         0.346
             100    1030301    6000000         0.472
             120    1771561   10368000         0.686
             140    2803221   16464000         0.983
             160    4173281   24576000         1.328
```

6.1 Tessellation by orthotopes

```
from fc_hypermesh import  bench
bench(2,range(1000,6000,1000),type='orthotope',box=[[-1,1],[-1,1]],order=1)
```

Listing 19: Tessellation of $[-1,1]^2$ by orthotopes

N	$\mathrm{n_q}$	$\mathrm{n_{me}}$	Python
1000	1 002 001	1 000 000	0.129
2000	4 004 001	4 000 000	0.327
3000	9 006 001	9 000 000	0.666
4000	16 008 001	16 000 000	1.12
5000	25 010 001	25 000 000	1.729

Table 7: Tessellation of $[-1,1]^2$ by orthotopes

```
from fc_hypermesh import  bench
bench(3,range(50,400,50),type='orthotope',box=[[-1,1],[-1,1],[-1,1]],order=1)
```

Listing 20: Tessellation of $[-1,1]^3$ by orthotopes

N	n_q	n_{me}	Python
50	132 651	125 000	0.139
100	1 030 301	1 000 000	0.269
150	3 442 951	3 375 000	0.536
200	8 120 601	8 000 000	1.058
250	15 813 251	15 625 000	1.89
300	27 270 901	27 000 000	3.132
350	43 243 551	42 875 000	4.824

Table 8: Tessellation of $[-1,1]^3$ by orthotopes

```
from fc_hypermesh import  bench
bench(4,[10,20,30,40,50,62],type='orthotope',order=1,
    box=[[-1,1],[-1,1],[-1,1],[-1,1]])
```

Listing 21: Tessellation of $[-1,1]^4$ by orthotopes

N	n_q	n_{me}	Python
10	14 641	10 000	0.206
20	194 481	160 000	0.294
30	923 521	810 000	0.399
40	2 825 761	2 560 000	0.717
50	6 765 201	6 250 000	1.488
62	15 752 961	14 776 336	3.053

Table 9: Tessellation of $[-1,1]^4$ by orthotopes

```
from fc_hypermesh import  bench
bench(5,[5,10,15,20,25,27],type='orthotope',order=1,
    box=[[-1,1],[-1,1],[-1,1],[-1,1],[-1,1]])
```

Listing 22: Tessellation of $[-1,1]^5$ by orthotopes

N	n_q	n_{me}	Python
5	7 776	3 125	0.383
10	161 051	100 000	0.467
15	1 048 576	759 375	0.751
20	4 084 101	3 200 000	1.562
25	11 881 376	9 765 625	3.955
27	17 210 368	14 348 907	5.535

Table 10: Tessellation of $[-1,1]^5$ by orthotopes

6.2 Tessellation by simplices

```
from fc_hypermesh import  bench
bench(2,range(1000,6000,1000),type='simplex',box=[[-1,1],[-1,1]],order=1)
```

Listing 23: Tessellation of $[-1,1]^2$ by simplices

N	n_q	n_{me}	Python
1000	1 002 001	2 000 000	0.199
2000	4 004 001	8 000 000	0.473
3000	9 006 001	18 000 000	0.916
4000	16 008 001	32 000 000	1.528
5000	25 010 001	50 000 000	2.348

Table 11: Tessellation of $[-1,1]^2$ by simplices

```
from fc_hypermesh import  bench
bench(3,range(40,190,20),type='simplex',box=[[-1,1],[-1,1],[-1,1]],order=1)
```

Listing 24: Tessellation of $[-1,1]^3$ by simplices

N	n_q	n_{me}	Python
40	68 921	384 000	0.188
60	226 981	1 296 000	0.26
80	531 441	3 072 000	0.348
100	1 030 301	6 000 000	0.473
120	1 771 561	10 368 000	0.684
140	2 803 221	16 464 000	0.957
160	4 173 281	24 576 000	1.324
180	5 929 741	34 992 000	1.918

Table 12: Tessellation of $[-1,1]^3$ by simplices

```
from fc_hypermesh import  bench
bench(4,[10,20,25,30,35,40],type='simplex',order=1,
    box=[[-1,1],[-1,1],[-1,1],[-1,1]])
```

Listing 25: Tessellation of $[-1,1]^4$ by simplices

N	n_q	n_{me}	Python
10	14 641	240 000	0.276
20	194 481	3 840 000	0.555
25	456 976	9 375 000	0.98
30	923 521	19 440 000	1.768
35	1 679 616	36 015 000	3.039

Table 13: Tessellation of $[-1,1]^4$ by simplices

```
from fc_hypermesh import  bench
bench(5,range(2,14,2),type='simplex',order=1,
    box=[[-1,1],[-1,1],[-1,1],[-1,1],[-1,1]])
```

Listing 26: Tessellation of $[-1,1]^5$ by simplices

N	n_q	n_{me}	Python
2	243	3 840	0.42
4	3 125	122 880	0.41
6	16 807	933 120	0.556
8	59 049	3 932 160	0.794
10	161 051	12 000 000	1.449
12	371 293	29 859 840	3.079

Table 14: Tessellation of $[-1,1]^5$ by simplices

6 References

[1] F. Cuvelier. Vectorized algorithms for regular and conforming tessellations of d-orthotopes and their faces with high-order orthotopes or simplicial elements, March 2019. working paper or preprint.

[2] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[3] Matplotlib team. Matplotlib, 2017. User's Guide.

Informations for developpers/maintainers of the fc hypermesh Python package

git informations on the packages used to build this manual

```
  name: fc-hypermesh
   tag: 0.1.1
commit: 97eb31268a3ddef0095d487bcb11190a309529f0
  date: 2019-12-29
  time: 14-51-49
status: 0


  name: fc-bench
   tag: 0.2.0
commit: 56ba4901391836cc91e2ce64c6f15699b966fdd5
  date: 2019-12-23
  time: 07-53-49
status: 0


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

git informations on the LaTeX package used to build this manual

```
  name: fctools
   tag:
commit: c73b7a2fbc6bfb1a5daf17097463a3a6f3541282
  date: 2019-03-22
  time: 12-57-26
status: True
```

git informations on the fc_config package used to build this distribution

```
  name: fc-config
   tag:
commit: 5333e20f025b614ae3d91a31521d38edfee6629c
  date: 2019-12-30
  time: 07-31-28
status: True
```