2023-06-19_010_Meshes_Part-01

June 19, 2023

[1]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline

from matplotlib.animation import FuncAnimation
matplotlib.rc('animation',html='html5')
import colorcet as ccm
from graphplot import *
```

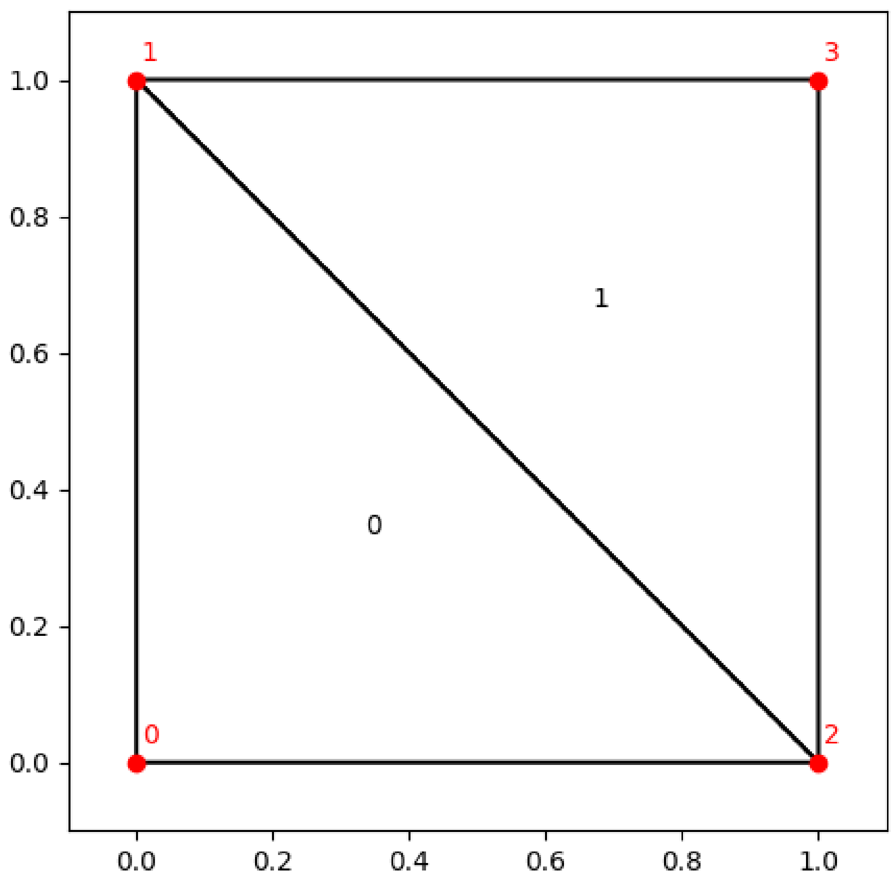
[2]:
```
import FEM
```

0.1 Points and triangles

[3]:
```
# specify domain by giving its extremal points
points=np.array([[0,0],[0,1],[1,0],[1,1]],dtype=np.double)
dim=points.shape[1]

# manually create simple triangulation
pointData=points.copy()
triangleData=np.array([[1,0,2],[2,3,1]],dtype=np.int32)
nPoints=pointData.shape[0]
nTriangles=triangleData.shape[0]
```

[4]:
```
fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
ax.scatter(pointData[:,0],pointData[:,1],c="r")
```

```
# annotate points
for i,x in enumerate(pointData):
    ax.annotate(i,x,(5,5),textcoords="offset␣
↪points",va="bottom",ha="center",color="r")

for i,t in enumerate(triangleData):
    cycleData=np.concatenate((t,t[[0]]))
    plt.plot(pointData[cycleData,0],pointData[cycleData,1],c="k",zorder=-1)
    center=np.mean(pointData[t],axis=0)
    ax.annotate(i,center,color="k")

buffer=0.1
plt.xlim([-buffer,1+buffer])
plt.ylim([-buffer,1+buffer])
plt.tight_layout()
plt.show()
```



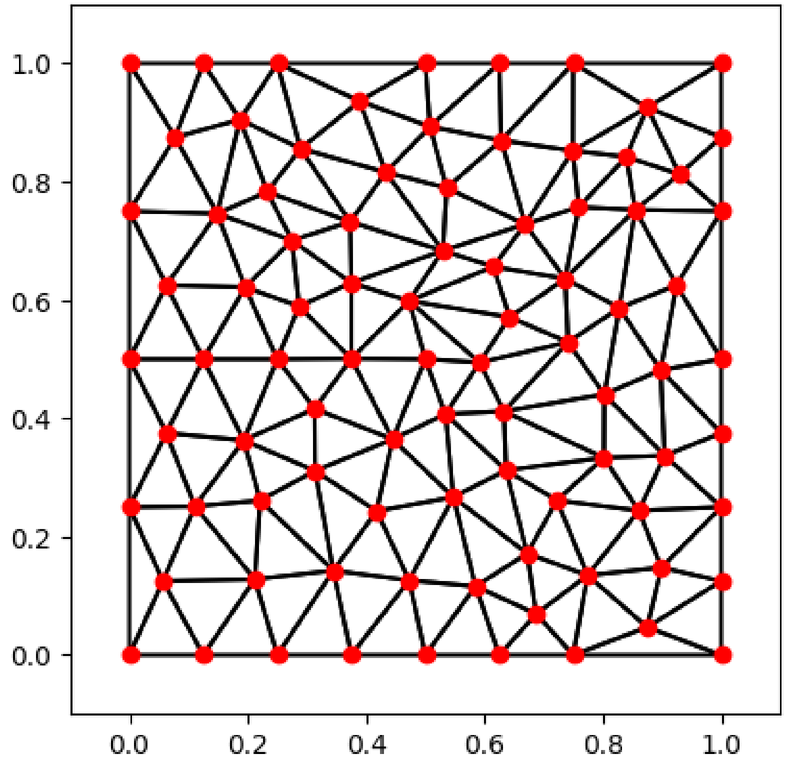
[5]:
```
triangleData
```

[5]:
```
array([[1, 0, 2],
       [2, 3, 1]], dtype=int32)
```

0.1.1 Load a finer triangulation

[5]:
```python
# create/load a finer example mesh
if False:
    import lib.Triangle as Triangle

    pointData,triangleData=Triangle.createMesh(points,maxArea=0.01)
    np.savez("triangulation.npz",pointData=pointData,triangleData=triangleData)
else:
    data=np.load("triangulation.npz")
    pointData=data["pointData"]
    triangleData=data["triangleData"]
    data.close()
nPoints=pointData.shape[0]
nTriangles=triangleData.shape[0]
```

[7]:
```python
fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
ax.scatter(pointData[:,0],pointData[:,1],c="r")

for i,t in enumerate(triangleData):
    cycleData=np.concatenate((t,t[[0]]))
    plt.plot(pointData[cycleData,0],pointData[cycleData,1],c="k",zorder=-1)

buffer=0.1
plt.xlim([-buffer,1+buffer])
plt.ylim([-buffer,1+buffer])
plt.show()
```



[8]:
```
nTriangles
```

[8]: 150

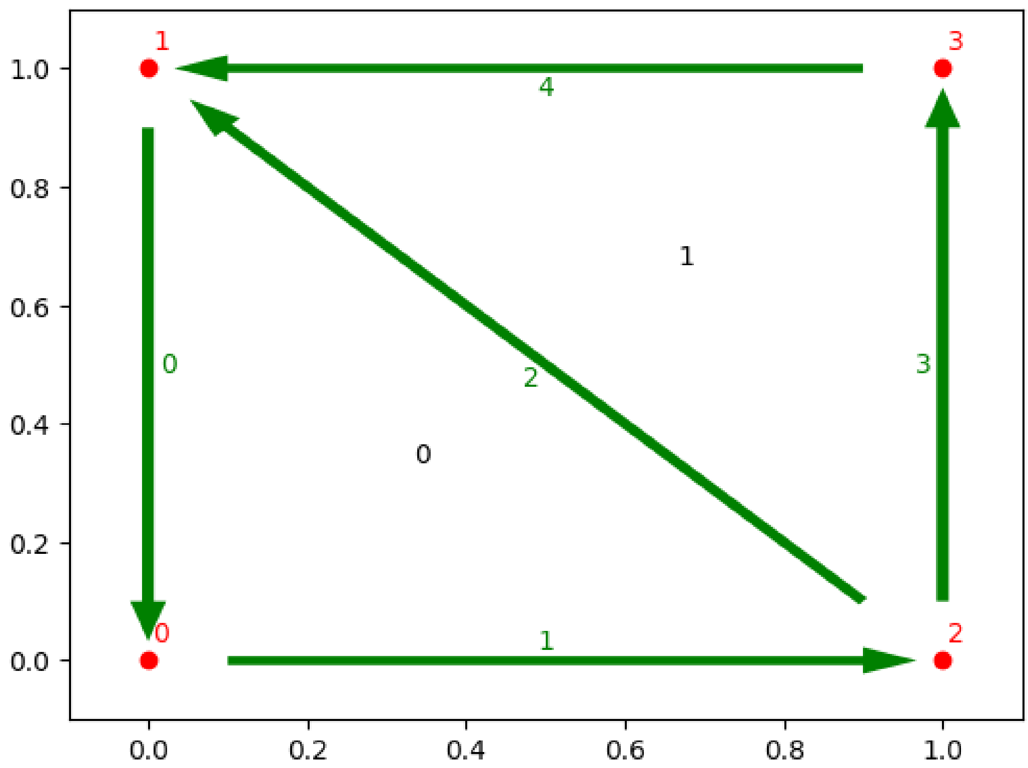
0.2 Edges

[9]:
```
# note: currently we draw every edge twice, as part of the two adjacent␣
↪triangles
# it is almost always useful to explicitly extract a list of edges
# and the relation between edges and triangles
```

[10]:
```
# demonstrate this on simple triangulation
pointData=points.copy()
triangleData=np.array([[1,0,2],[2,3,1]],dtype=np.int32)
nPoints=pointData.shape[0]
nTriangles=triangleData.shape[0]
```

[11]:
```
# extract edge data
edgeData,etAdjacencyData,etAdjacencyDataOrientation=FEM.getEdges(triangleData)
nEdges=edgeData.shape[0]
```

[12]:
```
# draw mesh with annotated edges
fig=plt.figure()
ax=fig.add_subplot()
ax.scatter(pointData[:,0],pointData[:,1],c="r")
# annotate points
for i,x in enumerate(pointData):
    ax.annotate(i,x,(5,5),textcoords="offset points",va="bottom",ha="center",color="r")

# annotate triangle centers
for i,t in enumerate(triangleData):
    center=np.mean(pointData[t],axis=0)
    ax.annotate(i,center,color="k")

# draw and annotate edges
for i,e in enumerate(edgeData):
    x=pointData[e[0]]
    y=pointData[e[1]]
    # implement some manual shrinking
    shrink=0.1
    a=(1-shrink)*x+shrink*y
    delta=(1-2*shrink)*(y-x)

    ax.arrow(a[0],a[1],delta[0],delta[1],width=0.015,lw=0,color="g")

    orientation=delta
    flip=[-orientation[1],orientation[0]]
    flip=flip/np.linalg.norm(flip)
    midpoint=0.5*x+0.5*y
    ax.annotate(i,midpoint,10*flip,textcoords="offset pixels",ha="center",va="center",c="g")
buffer=0.1
plt.xlim([-buffer,1+buffer])
plt.ylim([-buffer,1+buffer])
plt.show()
```



[13]:
```
# edge data: list of all edges in the triangulation
# as for graphs: each edge is given by indices of two connected points
edgeData
```

[13]:
```
array([[1, 0],
       [0, 2],
       [2, 1],
       [2, 3],
       [3, 1]], dtype=int32)
```

[14]:
```
# triangle-edge adjacency data:
# for each triangle: what are the indices of the edges that represent its sides
etAdjacencyData
```

[14]:
```
array([[0, 1, 2],
       [3, 4, 2]], dtype=int32)
```

[15]:
```
# orientation data:
# each triangle has an orientation (counter-clockwise)
# is the orientation of a boundary edge aligned (+1) or opposite (-1) ?
etAdjacencyDataOrientation
```

[15]: array([[1, 1, 1],
        [1, 1, -1]], dtype=int32)

0.3 Plotting functions

[16]:
```
# again: create/load a finer example mesh
data=np.load("triangulation.npz")
pointData=data["pointData"]
triangleData=data["triangleData"]
data.close()
nPoints=pointData.shape[0]
nTriangles=triangleData.shape[0]
```

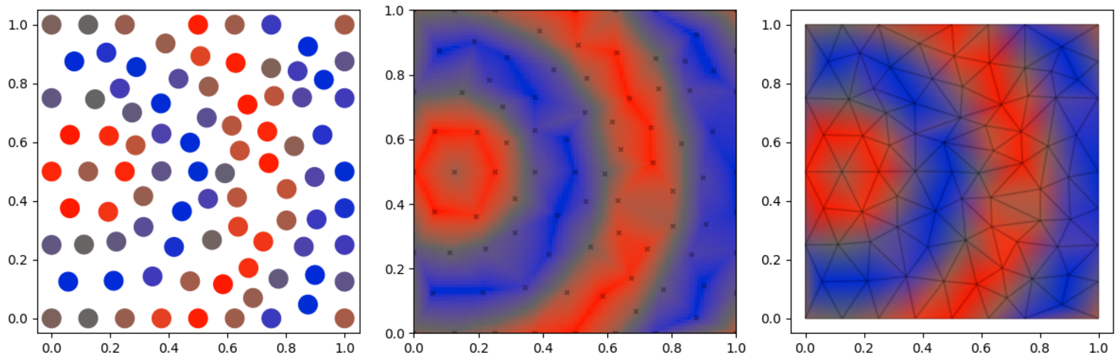
[17]:
```
# extract edge data
edgeData,etAdjacencyData,etAdjacencyDataOrientation=FEM.getEdges(triangleData)
nEdges=edgeData.shape[0]
```

[18]:
```
# compute some function on the vertices
x=pointData[:,0]
y=pointData[:,1]

f=np.sin(((x-0.1)**2+(y-0.5)**2)**0.5*2*np.pi*2)
vmin=-1
vmax=1
f=(f-vmin)/(vmax-vmin)
colfun=ccm.cm.CET_D8
```

0.3.1 Plotting in 2d, with color

[19]:
```
fig=plt.figure(figsize=(12,4))

ax=fig.add_subplot(1,3,1,aspect=1.)
ax.scatter(pointData[:,0],pointData[:,1],c=colfun(f),s=200)

ax=fig.add_subplot(1,3,2,aspect=1.)
nPts=100
xGrid=yGrid=np.linspace(0,1,num=nPts)
XGrid,YGrid=np.meshgrid(xGrid,yGrid)
dat=scipy.interpolate.griddata((pointData[:,0],pointData[:,1]),f,(XGrid.
 ↪ravel(),YGrid.ravel()),method="linear")
ax.imshow(dat.reshape((nPts,nPts)),extent=(0,1,0,1),origin="lower",cmap=colfun)
ax.scatter(pointData[:,0],pointData[:,1],color=(0,0,0,0.3),marker="x",s=10)
```

```
ax=fig.add_subplot(1,3,3,aspect=1.)
tri=matplotlib.tri.Triangulation(pointData[:,0],pointData[:,1],triangleData)
ax.tripcolor(tri,f,shading='gouraud',cmap=colfun)
lineCollection=matplotlib.collections.
 ↪LineCollection(pointData[edgeData],zorder=2,color="k",alpha=0.2)
ax.add_collection(lineCollection)


plt.tight_layout()
plt.show()
```

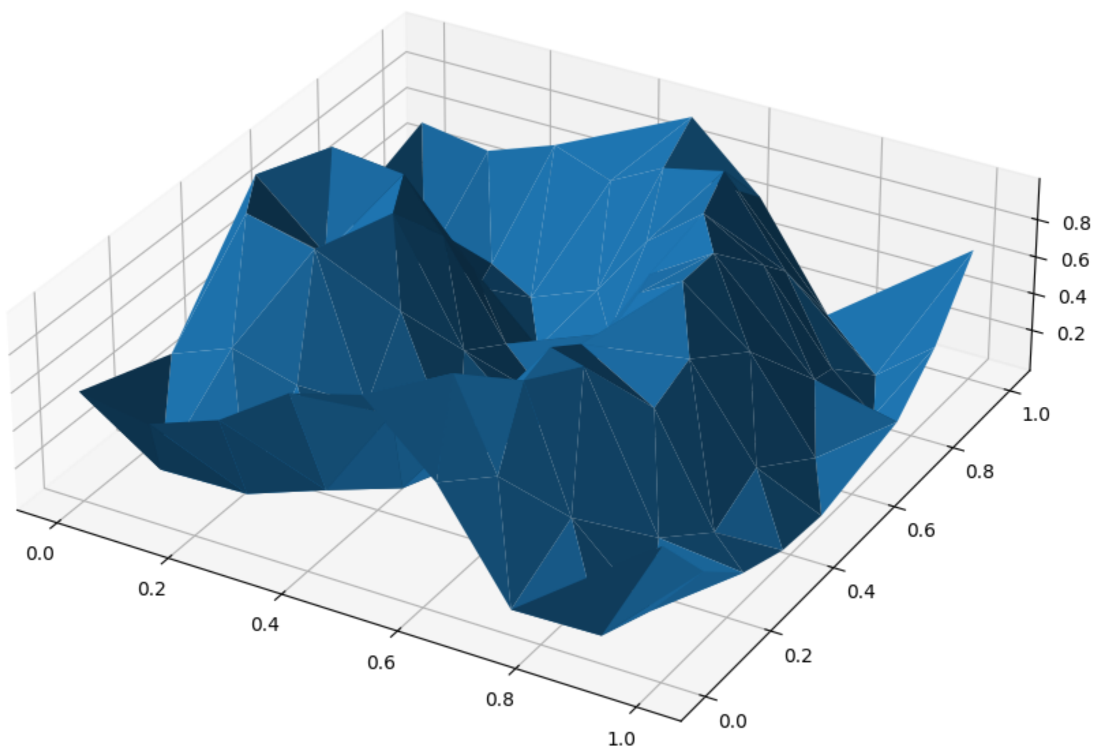


[20]:
```
# recall the interpolation example from the scatter plot session:
# this is one of the ways to handle this, if mesh information is available
```

0.3.2 Plotting in 3d

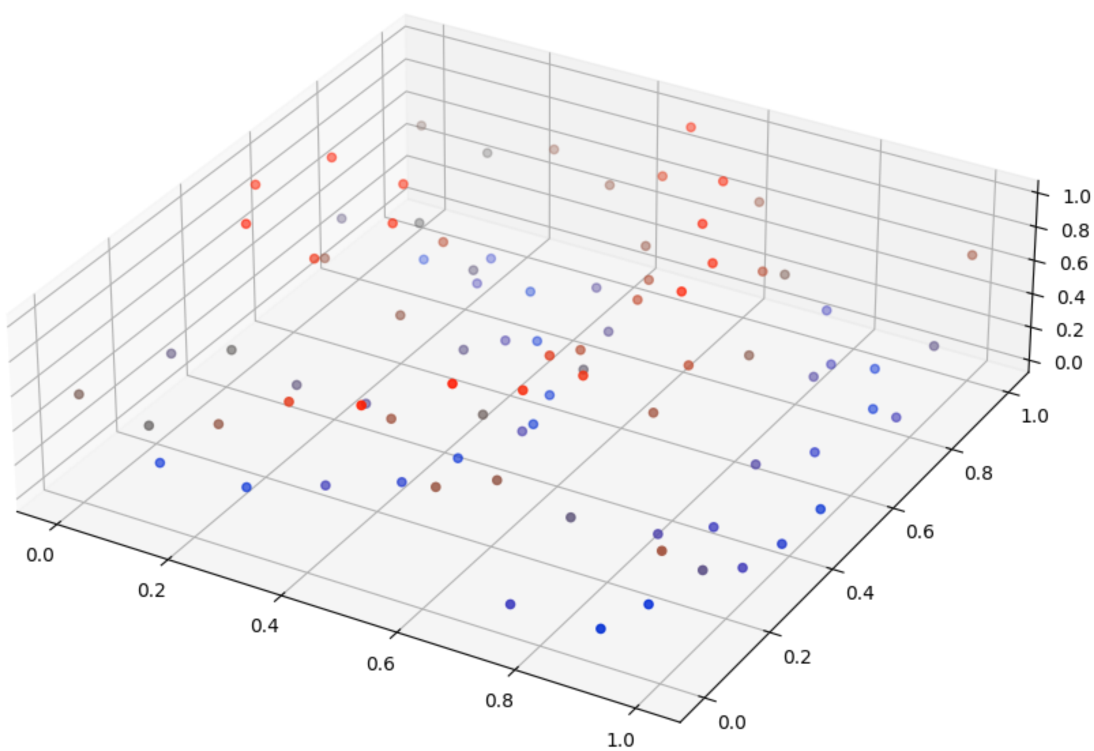
[21]:
```
%matplotlib inline
fig = plt.figure(figsize=(8,8))

ax = fig.add_subplot(projection='3d')

# uniform color with shading
ax.plot_trisurf(pointData[:,0], pointData[:,1], f, triangles=triangleData)

# color by height
#   only works approximately, since matplotlib can only assign one color to␣
 ↪each triangle
#   smooth color interpolation within each triangle is not available in 3d, but␣
 ↪is available in many
#   other plotting libraries / environments
#   also, matplotlib cannot do light-shading with variable colors, so we lose a␣
 ↪lot of the 3d impression
```

```
#   other plotting libraries are more powerful in this respect
#ax.plot_trisurf(pointData[:,0], pointData[:,1], f, triangles=triangleData,␣
 ↪cmap=colfun)

ax.set_box_aspect((1,1,0.3))

plt.tight_layout()
plt.show()
```



[30]:
```
# for comparison: still much better than only the points
# adding triangles is the 2d/3d equivalent of adding lines in a scatter plot
%matplotlib inline
fig = plt.figure(figsize=(8,8))
```

```
ax = fig.add_subplot(projection='3d')

# uniform color with shading
ax.scatter(pointData[:,0], pointData[:,1], f, c=f,cmap=colfun)


ax.set_box_aspect((1,1,0.3))

plt.tight_layout()
plt.show()
```



[ ]: