2023-07-03_Interactive_01_Matplotlib_Animation

July 3, 2023

[1]:
```
import numpy as np
import scipy
import scipy.io as sciio
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from matplotlib.animation import FuncAnimation

import colorcet as ccm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
matplotlib.rc('animation',html='jshtml')

prop_cycle = plt.rcParams['axes.prop_cycle']
colors = prop_cycle.by_key()['color']

import colorcet as ccm
from graphplot import *
from bridge import *
%matplotlib inline
```

1 Matplotlib: animation (bridge example revisited)

1.1 Create the data for the animation

[2]:
```
# build graph
nRows=4
nCols=11
dim=2

nList=[nCols,nRows]
posList=getPoslistNCube((nCols,nRows),dtype=np.int32)
```

```
nPoints=posList.shape[0]

posList,edgeData=buildGridGraph2d(nRows,nCols,neighbourhood=8)

edgeLengths=np.linalg.norm(posList[edgeData[:,1]]-posList[edgeData[:,0]],axis=1)

L=getBridgeLaplacian(posList,edgeData,edgeLengths)
```

[3]:
```
V=np.zeros((nPoints,dim),dtype=np.double)
# apply a downward force on the upper middle point
V[nCols//2*nRows+nRows-1,1]=-1.
# keep vertices in first and last column fixed ("end points of bridge")
freeVertices=np.ones((nCols,nRows),dtype=bool)
freeVertices[0,:]=False
freeVertices[-1,:]=False
freeVertices=freeVertices.ravel()
nPointsFree=np.sum(freeVertices)
```

[4]:
```
deformation,force=getDeformation(L,V,freeVertices)
# rescale deformation for actual simulated deformation
mode=0.1*deformation.reshape((-1,dim))
```

1.2 Create actual animation

[5]:
```
# try to estimate maximal deformation for colorbar scaling
data=posList+mode
edgeLengthsNew=np.linalg.norm(data[edgeData[:,0]]-data[edgeData[:,1]],axis=1)
# difference of edge lengths
edgeLengthsDelta=edgeLengthsNew-edgeLengths
# set vmax to 0.5 of max abs value of this (so color scaling will saturate on␣
 ↪large deformations,
# but in exchange smaller ones also get some visibility)
vmax=0.5*np.max(np.abs(edgeLengthsDelta))

# choose a diverging colorbar from colorcet
colfun=ccm.cm.CET_D8
```

[6]:
```
mode.shape
```

[6]:
```
(44, 2)
```

[6]:
```
fig=matplotlib.figure.Figure()
ax=fig.add_subplot(aspect=1.)

# most plot commands return an object with which one can later manipulate the␣
 ↪displayed data
```

```
# for dynamic plot keep reference to these objects
pltobj_pts = ax.scatter([], [],c="k")
pltobj_lineCollection=matplotlib.collections.
 ↪LineCollection([[[0,0],[1,1]]],zorder=-1,lw=4)
ax.add_collection(pltobj_lineCollection)
ax.set_xlim([-0.5,nCols-0.5])
ax.set_ylim([-0.5,nRows-0.5])

def update(t):
    # this computes the deformation at time t, where a whole period lasts from␣
 ↪t=0 to t=1

    # compute updated locations of vertices
    data=posList+np.sin(2*np.pi*t)*mode

    # compute updated (deformed) edge lengths
    edgeLengthsNew=np.linalg.norm(data[edgeData[:,0]]-data[edgeData[:
 ↪,1]],axis=1)
    # difference to ground state edge lengths (which gives us stress or tension)
    edgeLengthsDelta=edgeLengthsNew-edgeLengths

    # compute new edge colors
    edgeColors=colfun(np.clip((1+edgeLengthsDelta/vmax)/2,0,1))

    # now update scatter-plot object with new vertex positions
    pltobj_pts.set_offsets(data)
    # update edge-plot object with line positions
    pltobj_lineCollection.set_paths(data[edgeData])
    # update edge-plot object with line colors
    pltobj_lineCollection.set_color(edgeColors)

# create animation object, arguments:
# * basic figure object to animate
# * update function to generate frames
# * arguments for calling update function
# * interval: time (in ms) per frame
ani = FuncAnimation(fig, update, frames=np.linspace(0, 1, 20,endpoint=False),
        blit=True,interval=2*1000/20)

ani
```

[6]:
```
<matplotlib.animation.FuncAnimation at 0x7fdd6f0a7880>
```

[ ]: