2023-06-19_001_Graphs_Basic

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
```

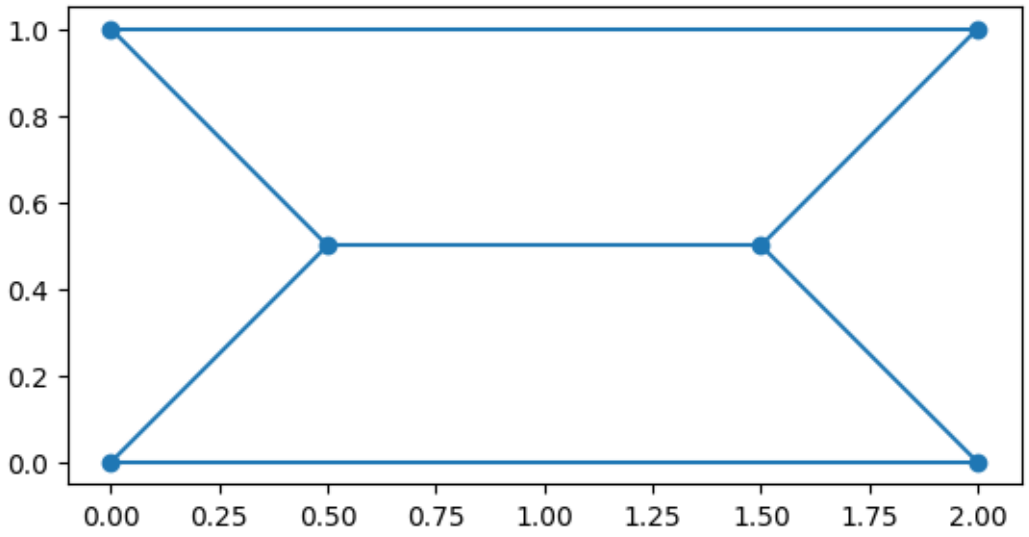
1 Graphs: getting started

1.1 Representing a graph and basic visualization

[2]:
```
# specify a simple (embedded) graph:
# * list of vertices with positions
# * list of (directed) edges, given as list of indices of involved vertices
#   convention: edge goes from first to second index
pointData=np.array([[0.,0.],[2.,0.],[0.5,0.5],[1.5,0.5],[0.,1.],[2.,1.
 ↪]],dtype=np.double)
edgeData=np.array([[0,1],[0,2],[1,3],[2,3],[2,4],[3,5],[4,5]],dtype=np.int32)

nPoints=pointData.shape[0]
nEdges=edgeData.shape[0]
```

[6]:
```
# basic display of graph: vertices and edges
fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
ax.scatter(pointData[:,0],pointData[:,1])
lineCollection=matplotlib.collections.
 ↪LineCollection(pointData[edgeData],zorder=-1)
```

```
ax.add_collection(lineCollection)
# usually introducing some convenience functions helps:
#vertices(ax,pointData)
#edges(ax,pointData,edgeData)
plt.tight_layout
plt.show()
```



[7]:
```
pointData[edgeData].shape
# 0: nr of edges
# 1: two vertices per edge
# 2: two coordinates per vertex
```

[7]: (7, 2, 2)

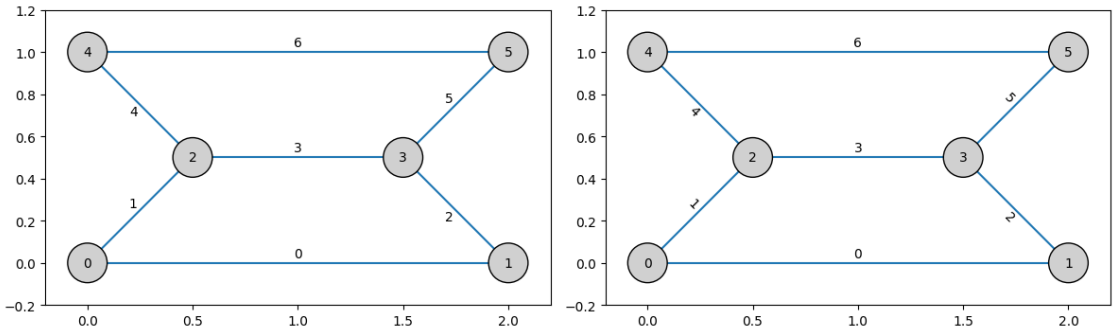
1.2 Simple graph and mesh plotting methods

[9]:
```
from graphplot import *
```

1.3 Annotations

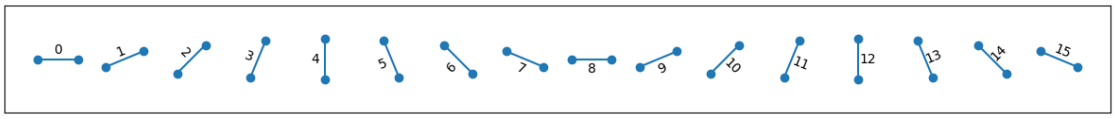
[10]:
```
# now with annotation
fig=plt.figure(figsize=(12,4))

for i,rotate in enumerate([False,True]):
    ax=fig.add_subplot(1,2,i+1,aspect=1.)
    edges(ax,pointData,edgeData)
```

```
    vertices(ax,pointData,annotate=True)
    annotateEdges(ax,pointData,edgeData,rotate=rotate)
    setBoxLimits(ax,pointData,buffer=0.2)

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



[11]:
```
# show annotation rotation for all edge rotations

# create point and edge data for individual edges at equally spaced different␣
  ↪orientations
nAngles=16
rad=0.3
angles=np.arange(nAngles)/nAngles*360
pointDataTmp=np.zeros((nAngles,2,2),dtype=np.double)
edgeDataTmp=np.zeros((nAngles,2),dtype=int)
for i,angle in enumerate(angles):
    pointDataTmp[i,0,0]=i-rad*np.cos(angle/180*np.pi)
    pointDataTmp[i,0,1]=-rad*np.sin(angle/180*np.pi)
    pointDataTmp[i,1,0]=i+rad*np.cos(angle/180*np.pi)
    pointDataTmp[i,1,1]=+rad*np.sin(angle/180*np.pi)
    edgeDataTmp[i,0]=2*i
    edgeDataTmp[i,1]=2*i+1
pointDataTmp=pointDataTmp.reshape((-1,2))

# visualize, to see how annotation orientation is adjusted
fig=plt.figure(figsize=(12,2))
ax=fig.add_subplot(aspect=1.)
edges(ax,pointDataTmp,edgeDataTmp)
vertices(ax,pointDataTmp,annotate=False)
annotateEdges(ax,pointDataTmp,edgeDataTmp,rotate=True,rotMode=45)
setBoxLimits(ax,pointDataTmp,buffer=0.5)
plt.xticks([])
plt.yticks([])
```

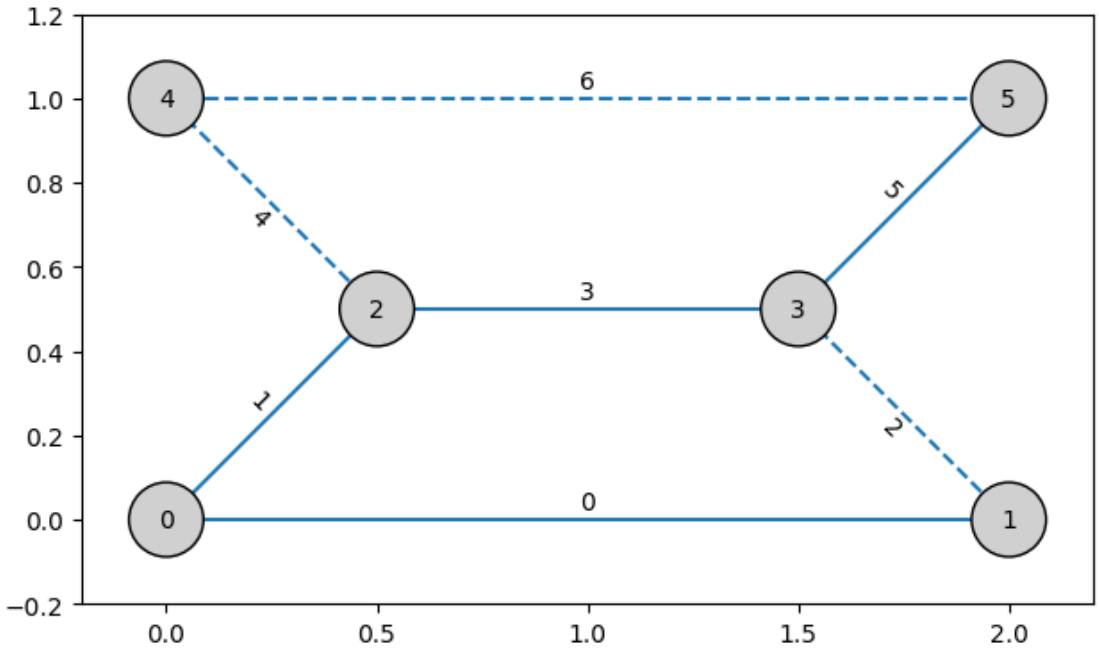
```
plt.tight_layout()
plt.show()
```



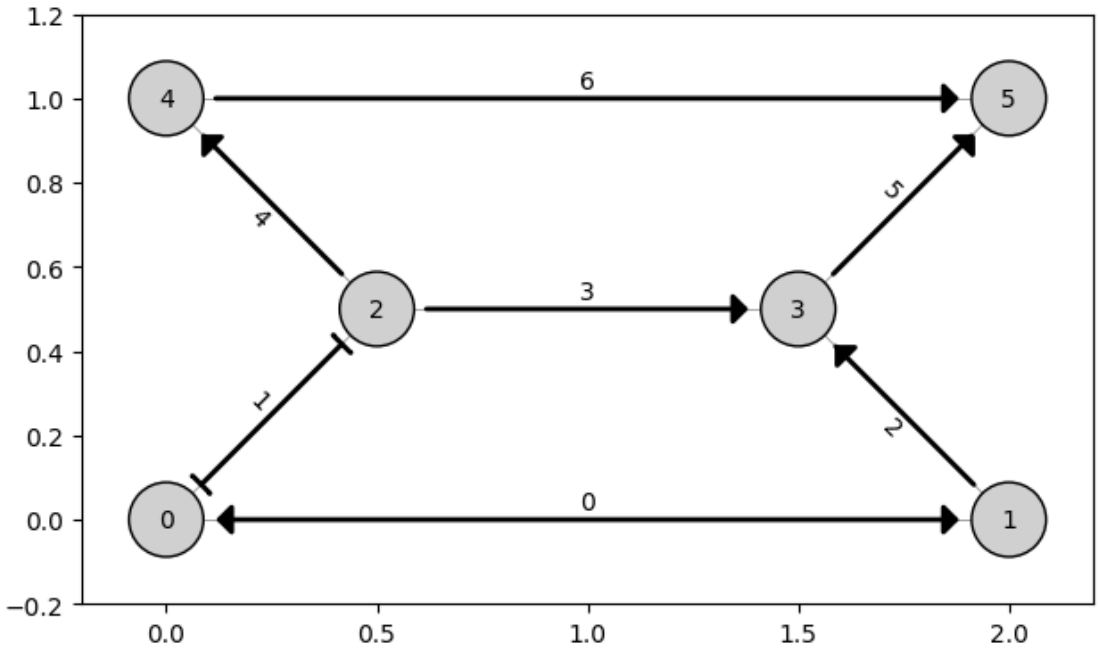
1.4 Edge-types and orientations

[12]:
```
# each edge now has a nominal "type", visualize these as different line styles
edgeTypeList=[0,0,1,0,1,0,1]
```

[13]:
```
# convert types to line styles
typeToLs=["solid","dashed"]
lsList=[typeToLs[i] for i in edgeTypeList]

fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
edges(ax,pointData,edgeData,ls=lsList)
vertices(ax,pointData,annotate=True)
annotateEdges(ax,pointData,edgeData,rotate=True)

setBoxLimits(ax,pointData,buffer=0.2)
plt.tight_layout()
plt.show()
```



[14]:
```
# specify arrow types for each edge, start with default "forward arrows"
orientationCodes=["-|>,head_length={0:s},head_width={0:s}" for i in␣
 ↪range(len(edgeData))]
# try a few different options
orientationCodes[0]="<|-|>,head_length={0:s},head_width={0:s}"
orientationCodes[1]="]-[,widthA={0:s},widthB={0:s}"

fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
vertices(ax,pointData,annotate=True,color="k")
edgesArrows(ax,pointData,edgeData,orientationCodes,shrink=20,size=5,color="k",lw=2)
annotateEdges(ax,pointData,edgeData,rotate=True)
# show full edge lines underneath, in faint style
edges(ax,pointData,edgeData,color="#808080",lw=0.5,zorder=-3)
setBoxLimits(ax,pointData,buffer=0.2)
plt.tight_layout()
plt.show()
```


1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.5
1.0
1.5
2.0
4
5
2
3
0
1
6
4
5
3
1
2
0

[ ]:

[ ]:

[ ]: