# 2023-06-19_002_Graphs_Flows

June 19, 2023

[3]:
```
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

import colorcet as ccm
from graphplot import *
```

## 1 Example: Flows on graphs

### 1.1 Background and simple example

- a common problem is to visualize a flow on a graph (could be a material, current, information, ...)
- flows are a signed (oriented) signal on the edges
- for each vertex, we can sum the total incoming and outgoing fluxes. This can be positive (mass increasing at the vertex), negative (mass decreasing) or zero (mass preserved). This is called the 'divergence' of the flow
- we will think a little bit about how to compute the divergence of a flow, and how to visualize the flow and its divergence

- let $(V, E)$ be a directed graph
- the following matrix is called **incidence matrix** of the graph:

$$A : \mathbb{R}^{|V|\times|E|}, \qquad A_{i,j} = \begin{cases} 1 & \text{if edge } j \text{ leaves vertex } i, \\ -1 & \text{if edge } j \text{ enters vertex } i, \\ 0 & \text{else.} \end{cases}$$

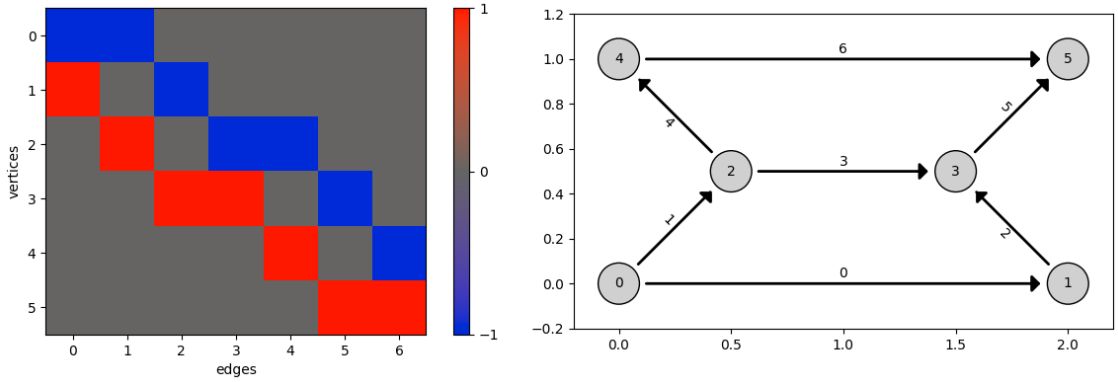
- now let $f \in \mathbb{R}^E$ be a flow on the graph where $f_j$ is the mass flowing on edge $j$,
- we adopt the convention that $f_j > 0$ if the mass is flowing **along** the orientation of edge $j$,
- and negative if it flows **against** the orientation.
- now assume we sit at vertex $i$, how much mass is flowing into and out of this vertex?
- we denote this by $m_i$ (positive if incoming, negative otherwise)
- if edge $j$ is incoming, then $f_j$ will contribute to incoming mass if it is positive, and to outcoming mass if it is negative. note that in this case we have $A_{i,j} = 1$.
- if edge $j$ is outgoing, then the signs are reversed; and in tis case $A_{i,j} = -1$.
- if edge $j$ is not connected to vertex $i$, then $A_{i,j} = 0$.
- so the total net amount of incoming mass is given by:

$$m_i = \sum_j A_{i,j} f_j = (Af)_i \quad \text{and therefore} \quad m = Af$$

[4]:
```
# specify a simple (embedded) graph:
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]
```

[5]:
```
# build the adjacency matrix
# note: only depends on edgeData, NOT on pointData
# in many cases this matrix is large, but sparse.
# then a more efficient implementation should be used
A=np.zeros((nPoints,nEdges))
A[edgeData[:,0],np.arange(nEdges)]=-1
A[edgeData[:,1],np.arange(nEdges)]=1
```

[6]:
```
fig=plt.figure(figsize=(12,4))
fig.add_subplot(1,2,1)
obj=plt.imshow(A,cmap=ccm.cm.CET_D8)
plt.xlabel("edges")
plt.ylabel("vertices")
plt.colorbar(obj,ticks=[-1,0,1])

ax=fig.add_subplot(1,2,2,aspect=1.)
orientationCodes=["-|>,head_length={0:s},head_width={0:s}" for i in␣
 ↪range(len(edgeData))]
edgesArrows(ax,pointData,edgeData,orientationCodes,shrink=20,size=5,color="k",lw=2)
vertices(ax,pointData,annotate=True)
annotateEdges(ax,pointData,edgeData,rotate=True)
setBoxLimits(ax,pointData,buffer=0.2)


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



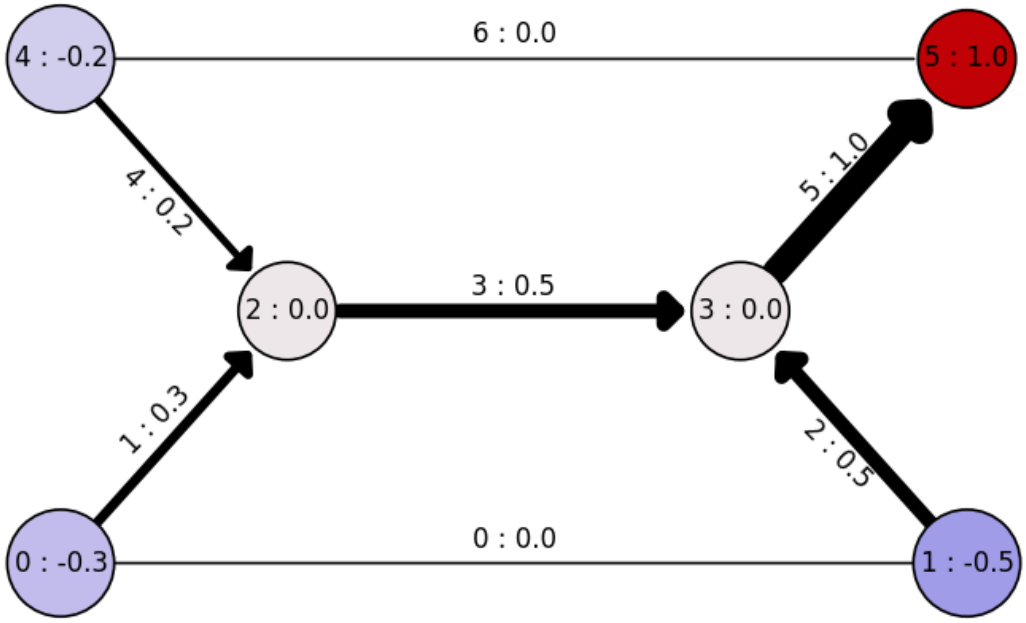

[7]:
```
# now define a flow vector f
f=np.zeros(nEdges,dtype=np.double)
f[1]=0.3
f[4]=-0.2
f[3]=0.5
f[2]=0.5
f[5]=1
```

[8]:
```
# use graph divergence operator to compute sources and sinks of flow
m=A.dot(f)
print(m)
```

```
[-0.3 -0.5  0.   0.  -0.2  1. ]
```

[9]:
```
# arrows and annotations
# now with annotation
fig=plt.figure(figsize=(6,4),dpi=120)
ax=fig.add_subplot()

flowThresh=1E-5
vmax=np.max(np.abs(f))
shrink=20
size=5
lwMax=10
lwMin=1


def getArrowCode(flow):
    if flow>flowThresh:
        return "-|>,head_length={0:s},head_width={0:s}"
    elif flow<-flowThresh:
        return "<|-,head_length={0:s},head_width={0:s}"
    else:
```

```
        return "-"

orientationCodes=[getArrowCode(flow) for flow in f]
#edgeColors=[ccm.cm.kr(flow/vmax) for flow in np.abs(f)]
#edgeColors=[ccm.cm.gray(0.5-0.5*flow/vmax) for flow in np.abs(f)]
edgeColors=[ccm.cm.gray(0) for flow in np.abs(f)]

for (x,y),o,col,flow in zip(pointData[edgeData],orientationCodes,edgeColors,f):
    a=matplotlib.patches.FancyArrowPatch(x,y,
            arrowstyle=o.format(str(size)),
            shrinkA=shrink, shrinkB=shrink,
            color=col,lw=(lwMax-lwMin)*np.abs(flow)/vmax+lwMin)
    ax.add_artist(a)


vmax=np.max(np.abs(m))
# annotate vertices
for i,(x,s) in enumerate(zip(pointData,m)):
    props = dict(boxstyle="circle,pad=0.25",fc=ccm.cm.CET_D1(0.5+0.5*s/vmax))
    ax.text(x[0],x[1],"{:d} : {:.1f}".
 ↪format(i,m[i]),ha="center",va="center",bbox=props,)

# annotate edges
msgList=["{:d} : {:.1f}".format(i,np.abs(f[i])) for i in range(nEdges)]
annotateEdges(ax,pointData,edgeData,textList=msgList,rotate=True,rotMode=90,dist=15)

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


4 : -0.2
6 : 0.0
5 : 1.0
4 : 0.2
5 : 1.0
2 : 0.0
3 : 0.5
3 : 0.0
1 : 0.3
2 : 0.5
0 : -0.3
0 : 0.0
1 : -0.5

[ ]: