In [ ]:
```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
import pygsp as pg
from pygsp import plotting
from scipy.linalg import expm
np.set_printoptions(precision=5)
np.set_printoptions(suppress=True)
```

We will try to do a simple example, let's take a delta for each node and evolve it with time

In [ ]:
```
N = 5 # number of nodes for our graph
G = pg.graphs.Path(N) # create path graph with N nodes
```

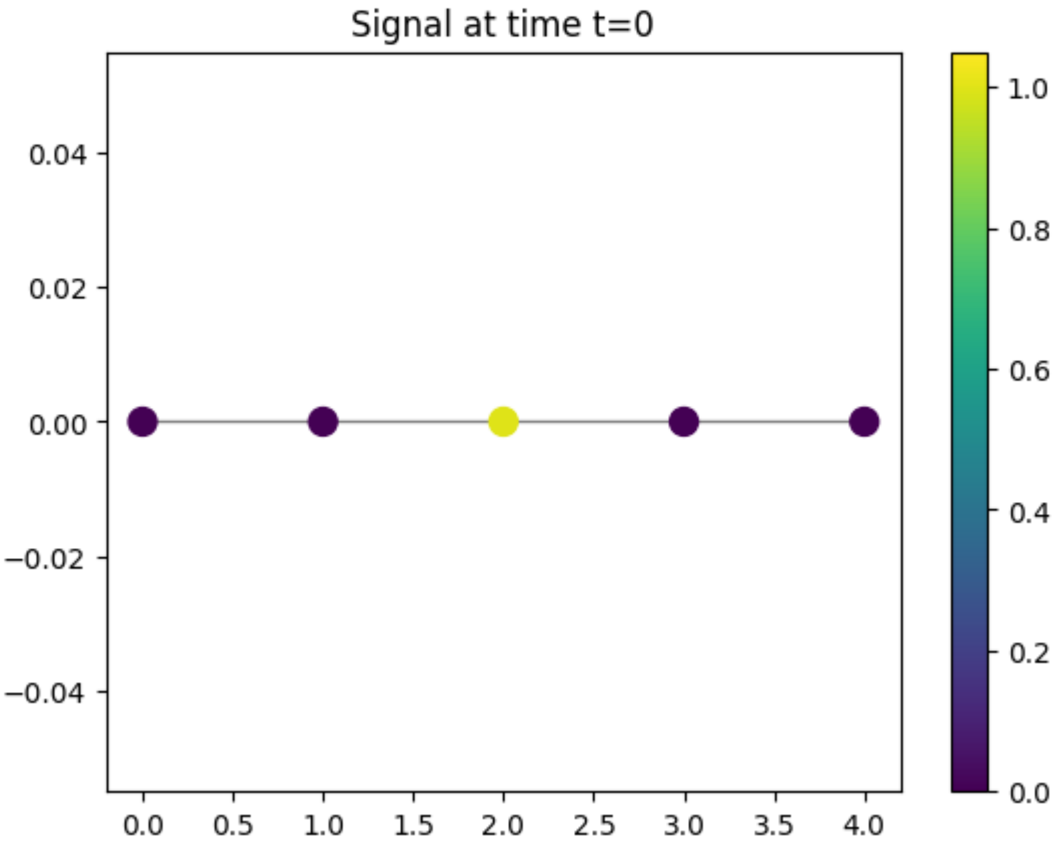
Deltas are the columns here

In [ ]:
```
H = np.eye(N)
```

Take third column and plot it

In [ ]:
```
plotting.plot_signal(G,H[:,2])
plt.title("Signal at time t=0")
```

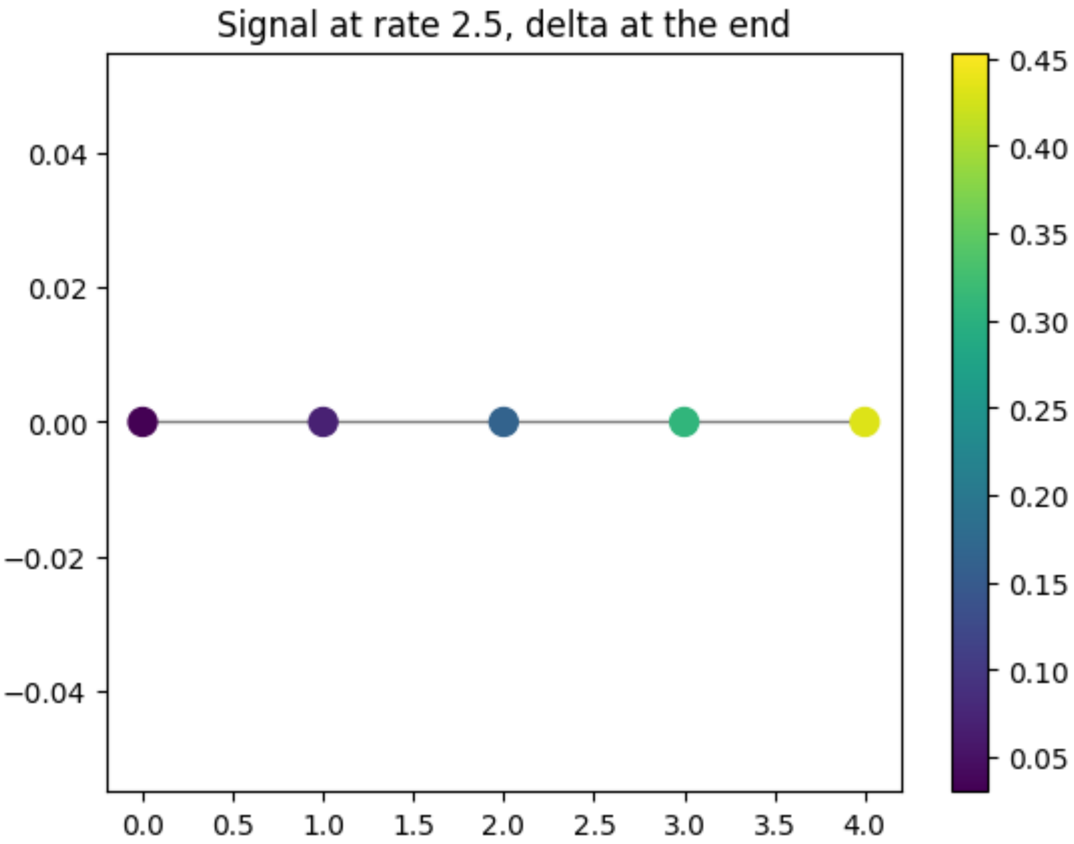
Out[ ]:
```
Text(0.5, 1.0, 'Signal at time t=0')
```



Let's evolve each signal $\tau = [1, 2.5, 4]$ units of time

In [ ]:
```
# extract the Laplacian from the line graph
L = G.L.toarray()
L = L/np.trace(L)*(L.shape[0])
# evolve the eye signal
signals = []
taus = [1,2.5,4]
for tau in taus:
    signals.append(expm(-tau*L)@H)
plotting.plot_signal(G,signals[1][:,4])
plt.title("Signal at rate 2.5, delta at the end")
```

Out[ ]:
```
Text(0.5, 1.0, 'Signal at rate 2.5, delta at the end')
```



Trying to learn from the diffused signal

In [ ]:
```
import learnHeat as lh
# concatenate created signals
X = np.concatenate([signal for signal in signals], axis=1)
# create random graph and Laplacian
M = X.shape[1]

_, L0, H0, tau0 = lh.create_signal(N=N,tau_ground=[1,2,3],M=M,se=0.1)
```

In [ ]:
```
L0
```

Out[ ]:
```
array([[ 2,  0,  0, -1, -1],
       [ 0,  1, -1,  0,  0],
       [ 0, -1,  2,  0, -1],
       [-1,  0,  0,  1,  0],
       [-1,  0, -1,  0,  2]])
```

In [ ]:
```
X.shape[:],L.shape[:],H.shape[:],len(taus) # H shape is wrong
```

Out[ ]: ((5, 15), (5, 5), (5, 5), 3)

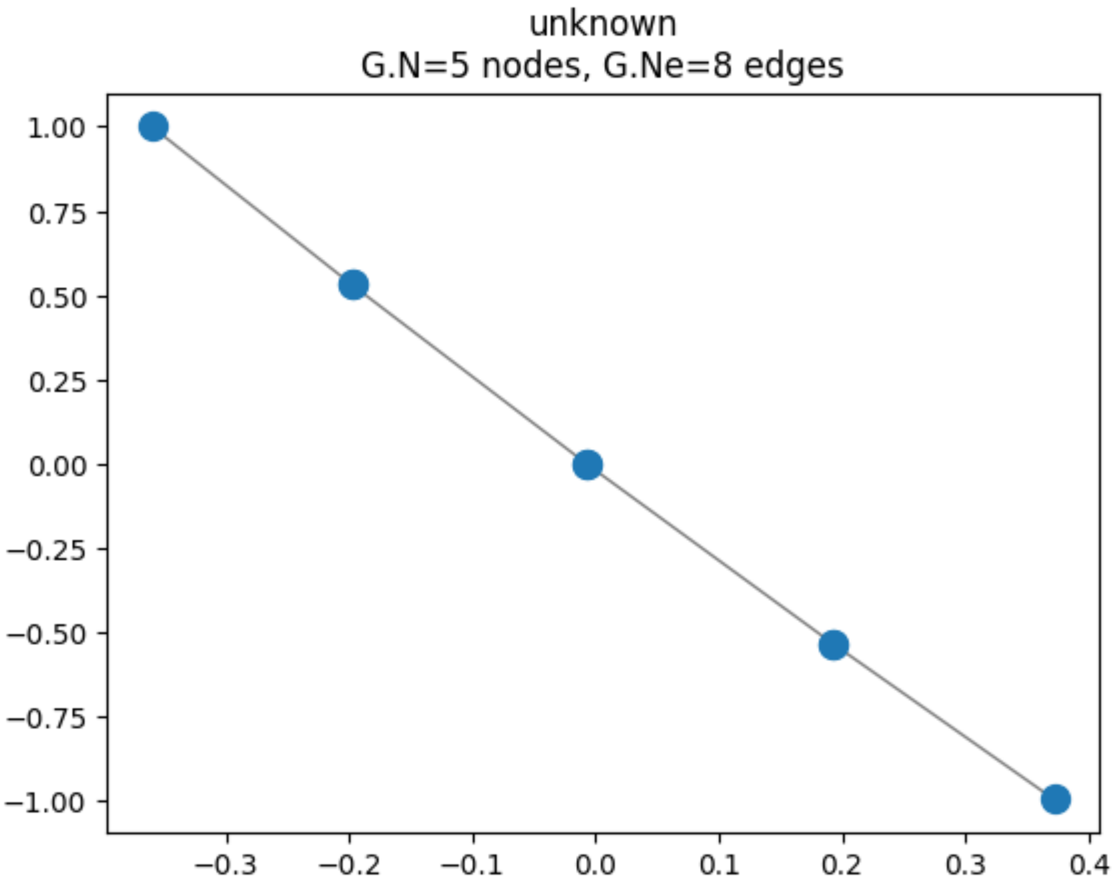
In [ ]:
```
result = lh.learn_heat(X = X,
                       L0=L, H0 = np.random.rand(15,15), tau0=[1,2,3],
                       verbose=False,
                       max_iter = 50, alpha = 0.1, beta=0.0)
```

```
Learning progress:   0%|          | 0/50 [00:00<?, ?it/s]
```

In [ ]:
```
result["L"]
```

Out[ ]:
```
array([[ 0.84497, -0.84497,  0.     ,  0.     ,  0.     ],
       [-0.84497,  1.10857, -0.2636 , -0.     ,  0.     ],
       [ 0.     , -0.2636 ,  0.41908, -0.15548,  0.     ],
       [ 0.     , -0.     , -0.15548,  1.39143, -1.23595],
       [ 0.     ,  0.     ,  0.     , -1.23595,  1.23595]])
```

In [ ]:
```
Lres = result["L"]
Lres[abs(Lres)<0.01] = 0
Adj = -np.copy(result["L"])
np.fill_diagonal(Adj, 0)
G_learned = pg.graphs.Graph(Adj)
G_learned.set_coordinates()
plotting.plot_graph(G_learned)
```

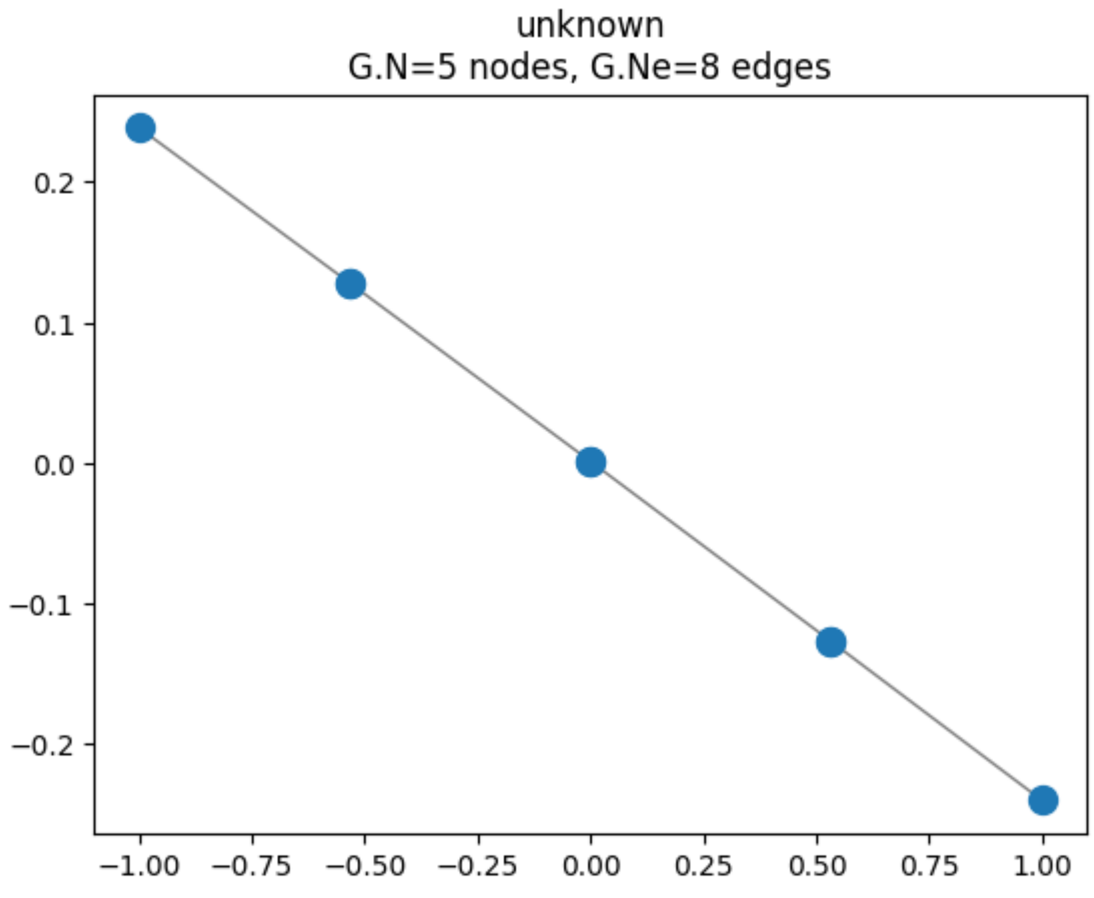


Now for the same graph, can we do the same technique with noise?

In [ ]:
```
se = 0.1 # standard deviation
noisyX = X + se*np.random.randn(X.shape[0],X.shape[1])
noisy_result = lh.learn_heat(X = X,
                    L0=L, H0 = np.random.rand(15,15), tau0=[1,2,3],
                    verbose=False,
                    max_iter = 50, alpha = 0.1, beta=0.01)
```

```
Learning progress:   0%|          | 0/50 [00:00<?, ?it/s]
```

In [ ]:
```
# do the same
noisy_Lres = result["L"]
noisy_Lres[abs(Lres)<0.01] = 0
noisy_Adj = -np.copy(result["L"])
np.fill_diagonal(noisy_Adj, 0)
noisy_G_learned = pg.graphs.Graph(noisy_Adj)
noisy_G_learned.set_coordinates()
plotting.plot_graph(noisy_G_learned)
```



Can we do the same with an arbitrary graph?

Let's create a random bigger graph with $N = 20$ as in the paper.

In [ ]:
```
from importlib import reload
lh = reload(lh)
big_N = 20
big_graph = nx.gnp_random_graph(big_N,p=0.4)
big_L = nx.laplacian_matrix(big_graph).toarray()
# we should normalize
big_L = big_L/np.trace(big_L)*big_N
big_tau = [0.5,1,1.5,2,3]
big_X = lh.create_deltas(big_L,big_tau,se=0)
```

Let's create false L, H, τ to feed the algorithm together with the noisy signal

In [ ]:
```
big_M = big_X.shape[1]
_, not_L, not_H, not_tau = lh.create_signal(N=big_N,M=big_M,tau_ground=bi
```

In [ ]:
```
big_res = lh.learn_heat(X=big_X,L0=not_L,H0=not_H,tau0=not_tau,alpha=0.05
```

```
Learning progress:   0%|          | 0/50 [00:00<?, ?it/s]
```

In [ ]:
```
big_L_learned = big_res["L"]
# create adjacency
big_W = -big_L_learned
np.fill_diagonal(big_W,0)
# cap with some threshold
big_W[abs(big_W)<0.01]=0
```

In [ ]:
```
big_tp = np.sum((big_W>0)&(big_L<0))
big_fp = np.sum((big_W>0)&(big_L>=0))
big_fn = np.sum((big_W<=0)&(big_L<0))
```

In [ ]:
```
precision = big_tp/(big_tp+big_fp)
```

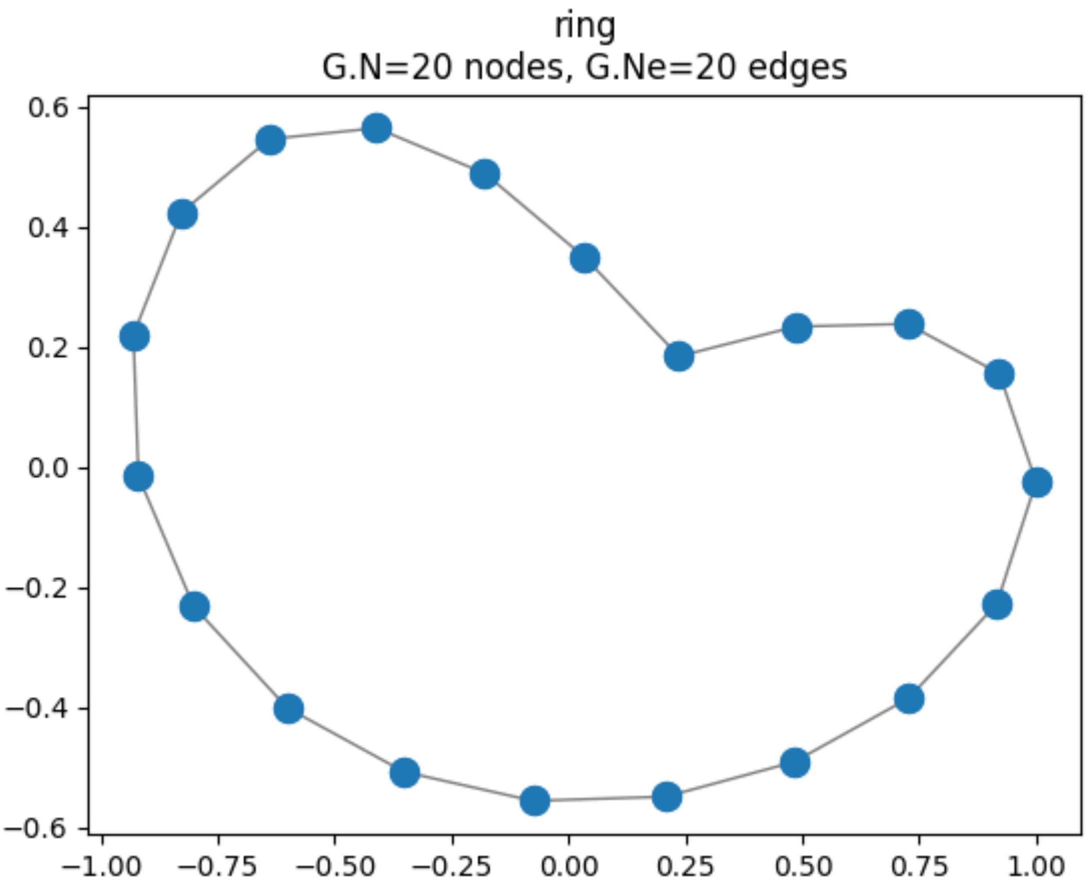
In [ ]:
```
recall = big_tp/(big_tp+big_fn)
```

In [ ]:
```
f_measure = 2/(1/precision + 1/recall)
f_measure
# we obtain 0.75 it is ok
```

Out[ ]:
```
0.7513227513227513
```

Finally, we will try to learn a ring!!!

In [ ]:
```
Nring = 20
Gring = pg.graphs.Ring(Nring,1)
Gring.set_coordinates()
plotting.plot_graph(Gring)
```



In [ ]:
```
Lring = Gring.L.toarray().asfloat()
Xring = lh.create_deltas(Gring.L,[0.5,1,1.5,5])
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[117], line 1
----> 1 Lring = Gring.L.toarray().asfloat()
      2 Xring = lh.create_deltas(Gring.L,[0.5,1,1.5,5])

AttributeError: 'numpy.ndarray' object has no attribute 'asfloat'
```