AMATH 563 Networks

Tyler Chen

ABSTRACT

We take connectivity graphs of neurons in the c. elegans worm and analyze the connectivity. This is done first using a graph theoretic approach, and then an approach based on the Kuramoto Oscillator.

I INTRODUCTION AND OVERVIEW

Networks arise naturally in a variety of applications. Being able to understand the behavior of large networks is of clear interest. In this report we compute degree and flow distributions of the adjacency matrix for the chemical synaptic network (directed) and adjacency matrix for the gap junction network (undirected). We then compute the diameter of the communication classes of each of these networks as well as the combination. We also put these graphs into the Kuramoto oscillator model and observe how the coupled the resulting systems are.

II THEORETICAL BACKGROUND

Graphs

We provided a basic introduction to the definitions used in graph theory [2].

Given some vertex set V, an undirected graph is the tuple $G=(V,E)$, where the edge set E contains elements of the form $e=\{u,v\}$ where $u,v\in G$.

The degree of a node $v\in V$ is the number of edges in E containing v. The flow of a node v is the sum of the weights of the edges containing v.

A subgraph G' of G is a graph formed by taking a subsets $V'\subseteq V$ and $E'\subseteq E$ such that every endpoint of E' is in V'.

The subgraph of G induced by V' is the graph $G'=(V',E')$ where $E'=\{\{u,v\}:u,v\in V'\}$.

A u-v path is a list of the form $p=(u=v_0,v_1,v_2,\ldots,v_k=v)$ such that $\{v_j,v_{j+1}\}\in E$ for $j=0,\ldots,k-1$. If such a path exists we say that v is reachable from u.

Clearly this gives gives rise to the notion of reachibility. That is, which vertices can be reached from which vertices. This forms an equivalence relation, where the equivalence classes are called the connected components of G.

We say the graph G is connected if there exists a u-v path between every pair of vertices $u,v\in V$. Equivalently, that there is only one connected component.

A weighting on the graph G is a function $w:E\to\mathbb{R}$. The weight of a $u-v$ path is the sum of the weights of the edges in this path.

A minimum length u-v path is a u-v path such that any other u-v path is longer (has more edges). A minimum weight u-v path is a u-v path such that any other u-v path has a larger weight.

The diameter of G is the length of the longest minimum length path between all pairs of vertices in V. If the graph is not connected the diameter is taken to be infinite.

Up to this point everything we have done has been relating to undirected graphs. Given some vertex set V, an undirected graph is the tuple $G=(V,E)$, where the edge set E contains elements of the form $e=(u,v)$ where $u,v\in G$. We now note that $(u,v)\in E$ does not imply $(v,u)\in E$.

Instead of the degree of a node v, we have the in-degree and the out-degree. Respectively these are the number of edges ending and starting at v. Similarly we have in-flow and out-flow.

A subgraph of a directed graph is defined in the same way as for an undirected graph. Similarly, a u-v path is defined in the same way.

However, reachability no longer forms an equivalence relation. Instead of connected components we can define communication classes to be the maximal subgraphs for which there is a path between any pair of nodes in the subgraph.

We say a directed graph is strongly connected if every vertex is reachable from every other vertex. Equivalently, that there is a single communication class. We say a directed graph is weakly connected if, treating all edges as undirected, the graph is connected.

Weighting on a directed graph is defined in the same way as for an undirected graph.

The diameter of directed graph is the longest minimum length path between all pairs of vertices in V. If a vertex is unreachable from any other vertex the diameter is take to be infinite.

Kuramoto Oscillator

The Kuramoto model describes the behavior of a large set of coupled oscillators [1].

Given N oscillators, their phases θ_i are given by the differential equation,

$$\frac{\mathrm{d}\theta_i}{\mathrm{d}t} = \omega_i + \frac{K}{N}\sum_{j=1}^{N} A_{ij}\sin(\theta_i - \theta_j) \tag{1}$$

where each oscillator as natural frequency w_i and K and A gives the coupling strengths.

In order to measure "how coupled" a system is. We define the order parameters r and ψ by,

$$re^{i\psi} = \frac{1}{N}\sum_{j=1}^{N} e^{i\theta_j} \tag{2}$$

where r is the phase coherence and ψ is the mean phase.

Note that $e^{i\theta_j}$ will be somewhere on the unit circle, with argument given by the phase (value of θ_j modulo 2π). If all the oscillators have a similar phase the mean of $e^{i\theta_j}$ will have a modulus close to one and argument close to the shared phase. Conversely, if the oscillators have completely different phases, the means of $e^{i\theta_j}$ will be somewhere near the origin.

Therefore, the value of r will give a measure of how similar the phases are at a given time. Taking the average value of r over some period of time after the oscillator has relaxed to a "steady state" will give a measure of how coupled the system is. Since $|e^{i\theta_j}| = 1$, $r = 1$ if all the phases are exactly the same. Otherwise, r will be smaller.

We can vary K and observe how r changes. Doing this for various A allows for the connectivity of various graphs to be compared. We know that as K increases the system will become increasingly coupled, however how quickly this happens will depend on how connected the oscillators are (sparsity of A) and other properties of the graph A.

III ALGORITHM IMPLEMENTATION AND DEVELOPMENT

All functions we use are taken from Scipy or core Python. In particular, we implement the Kuramoto Oscillator using scipy's `solve_ivp` and graph results are done using Scipy's `sparse.csgraph` module. While our graphs are not sparse this was the most easily accessible module with functions to compute the number of connected components of a graph as well as the distance between any two nodes.

Representing Graphs

Graphs are represented using weighted adjacency matrices. To compute quantities such as diameter we convert the weighted adjacency graphs to unweighted ones, where the i, j entry is `True` if there is an edge from i to j and `False` otherwise.

To convert an directed unweighted adjacency graph G to an undirected one we simply take $G \oplus G^T$ where $\oplus$ indicates boolean `or` applied componentwise to G and G^T.

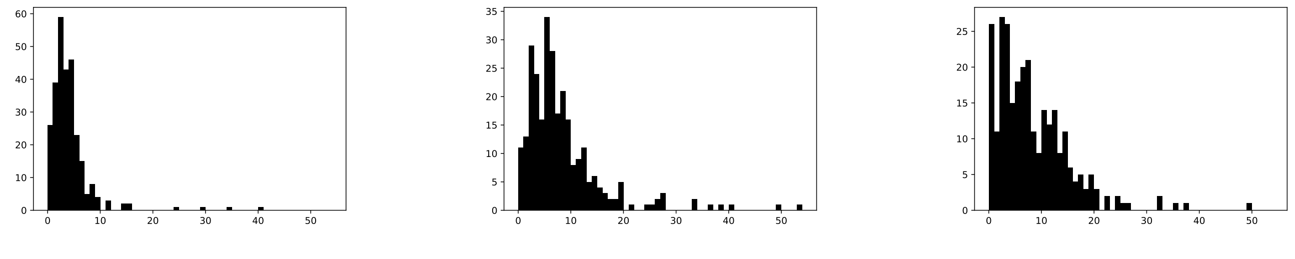
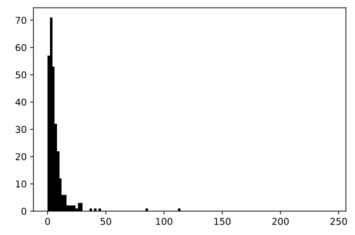
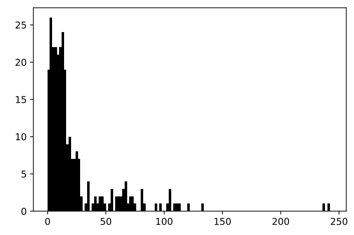

(a) L degree distribution (b) A_c in-degree distribution (c) A_c out-degree distribution

Figure 1: Degree Distributions (horizontal: degree, vertical: count)


(a) L flow distribution


(b) A_c in-flow distribution

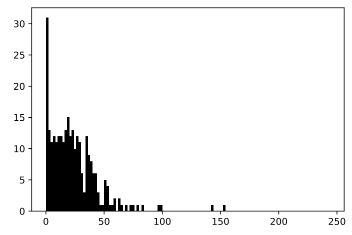

(c) A_c out-flow distribution

Figure 2: Flow Distributions (horizontal: flow, vertical: count)

Degree Distribution

The in-degree and out-degree of a vertex can be found by counting the number of nonzero entries in the corresponding column or row. In the case of an undirected graph these numbers are the same. Given a graph Laplacian we can also just take the diagonal entry.

Communication Classes

We can easily find which vertices belong to which communication class using `connected_components` and the appropriate keywords.

Diameter

We compute the diameter of a communication class by applying Dijkstra to each node pair and taking the maximum.

IV COMPUTATIONAL RESULTS

We are given data about the connectivity of neurons in the c. elegans worm. In particular, A_c is the adjacency matrix for the chemical synaptic network (directed) and L is the Laplacian matrix for the gap junction network (undirected). We immediately replace L with the adjacency matrix and from now on use L to denote the adjacency matrix rather than the graph Laplacian. The nodes of the two graphs are identical.

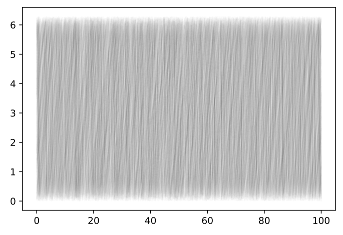
Note that since no background on what the weights of the graphs mean it is not necessarily meaningful to compare the graphs in this way. In fact, this was a major concern through out this process. Since we do not know whether a higher weight should at all correspond to a higher coupling, it is not clear that using the connections as the coupling makes any sense.

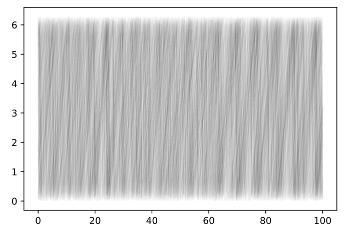
Graph Properties

Figure 1 shows the degree distributions of L and A_c. Similarly, Figure 2 shows the flow distributions of L and A_c. We immediately note that since both graphs have some nodes with degree zero that they are not connecte and so the diameters are infinite.

We next compute the connected components of L and the communication classes of A_c.

The undirected graph L has 29 connected components. The non-trivial components have 248, 3, and 2 nodes and have diameters 12, 2, and 1. The rest of the components are isolated nodes.


(a) $K = 0$


(b) $K = 10$

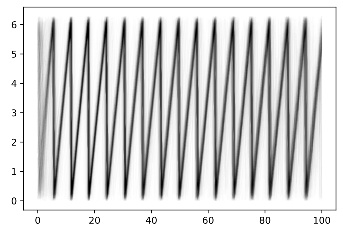
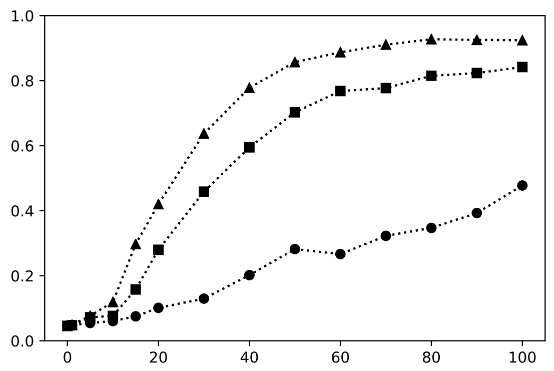
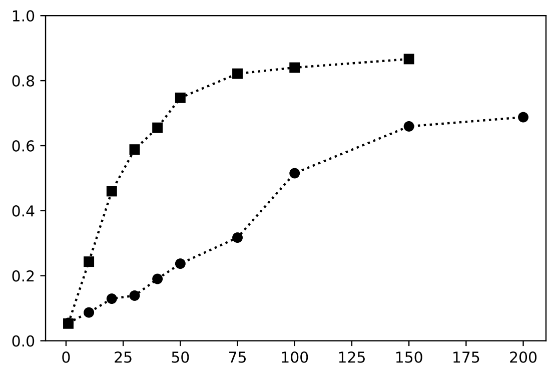

(c) $K = 50$

Figure 3: Coupling strength K (horizontal) vs. phases (vertical) for couplings given by C



(a) Unweighted adjacency graph



(b) Weighted adjacency graph

Figure 4: Coupling strength K (horizontal) vs. phase coherence r (vertical). circles: L, squares: A_c, triangles: C.

We note that A_c is weakly connected. The directed graph A_c has 42 communication classes. There non-trivial communication classes have 237 and 2 nodes and have diameters 10 and 1.

The diameter of A_c where all edges are treated as undirected is 6.

Since both L and A_c share the same nodes we combine them into a graph denoted C by combining the unweighted adjacency matrices component using a logical `or`. This means every edge of L or A_c is contained in C.

The resulting directed graph is weakly connected and has 6 communication classes. The one non-trivial communication class has 274 nodes and diameter 7.

The diameter of C treated as an undirected graph is 5.

We note that combining L and A_c significantly reduces the number of communication classes as well as the diameter of the largest communication class despite the fact taht the largest communication class of C has more node than those of A_c and L. This is not necessarily surprising as we have combined two graphs.

Kuramoto Oscillator

We now solve the Kuramoto Oscillator system. In particular, we pick natural frequencies randomly from a uniform distribution on $[1/2, 3/2]$ and random initial phases uniformly on $[0, 2\pi]$.

We iterate over various values of K and compute r for the unweighted and weighted adjacency graphs of L, A_c, and the combination graph C.

Figure 3 shows the phases of the oscillators over time at various values of K. Each oscillator has its path drawn with some opacity. Therefore, if most of the oscillators end up locked together the group phase should be clear. As expected, as K increases the phase locking becomes stronger.

Figure 4 shows the phase coherence as a function of the coupling strength for the unweighted adjacency graphs of L, A_c, and C. As expected, as K increases the phase coherence increases for all graphs. Moreover, the more connected a graph is the lower the value of K has to be to reach a fixed coherence value.

While there is a slight difference between Figures 4a and 4b, the general behavior is the same. The most notable differences are near the origin, where the unweighted adjacency graphs have a much slower increase in coupling. We would have liked to increase K even further, however some of the data points took over an hour to generate. Given more time it would be trivial extend these plots.

Since it is not clear what the objective of this model is we do not continue this further. However, we note that no matter how large the coupling strength, disconnected components will not couple together. This means that there could be multiple coupled groups traveling independently from one another. That said, we do not need the oscillator model to determine that the graphs have connected components.

V SUMMARY AND CONCLUSIONS

We compute degree and flow distributions of the adjacency matrix for the chemical synaptic network (directed) and adjacency matrix for the gap junction network (undirected). We then compute the diameter of the communication classes of each of these networks as well as the combination. Since we have no physical understanding of what our data means, we look which nodes are connected rather than the strength of these connections.

We put these graphs into the Kuramoto oscillator model and are able to observe how the phase coherence increases as the coupling strength increases. In particular we notice that the more connected the graphs are the quicker they become coupled. For instance, since C is a supergraph of L and A_c, it is not surprising that the Kuramoto oscillator with coupling described by C becomes coherent at a lower value of K than the oscillators corresponding to either K or A_c.

Overall we found it very unclear what the purpose of computing any of these quantities was. Since only very limited information was given about what L and A_c represent it is not clear that we can combine the data in any meaningful way. More specifically, if we knew that the weights represented the strength of connection, or time for a signal to travel the connection, etc, then a distance metric could be defined for the graphs and we could discuss physical quantities such as how long it would take a signal to travel between two nodes, etc. However, without knowing this we are reduced to observing topological properties of the graphs.

REFERENCES

[1] J. Nathan Kutz. Course lecture notes. 2018.

[2] Alexander Shrijver. A course in combinatorial optimization. 2009.

VI APPENDIX A

All functions are included in the code in Appendix B as they are specific to the task performed in each of the files.

VII APPENDIX B

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May 30 17:41:35 2018

@author: tyler
"""

import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
from scipy.io import loadmat
from scipy.sparse import csgraph


prams = sp.io.loadmat('prams.mat')

L = prams['L']
Ac = prams['Ac']

#%% set up weighted adjacency graphs
L_g = np.diag(np.diag(L)) - L
Ac_g = Ac


#%% compute degree distribution
L_deg = np.count_nonzero(L_g,axis=0)

Ac_deg_in = np.count_nonzero(Ac_g,axis=0)
Ac_deg_out = np.count_nonzero(Ac_g,axis=1)

plt.figure()
plt.hist(L_deg,np.arange(np.max(55)),color='k')
plt.savefig('img/L_deg.pdf')

plt.figure()
plt.hist(Ac_deg_in,np.arange(np.max(55)),color='k')
plt.savefig('img/Ac_deg_in.pdf')

plt.figure()
plt.hist(Ac_deg_out,np.arange(np.max(55)),color='k')
plt.savefig('img/Ac_deg_out.pdf')

#%% compute flow distribution
L_flow = np.sum(L_g,axis=0)

Ac_flow_in = np.sum(Ac_g,axis=0)
Ac_flow_out = np.sum(Ac_g,axis=1)

plt.figure()
plt.hist(L_flow,np.arange(np.max(245),step=2),color='k')
plt.savefig('img/L_flow.pdf')

plt.figure()
plt.hist(Ac_flow_in,np.arange(np.max(245),step=2),color='k')
plt.savefig('img/Ac_flow_in.pdf')

plt.figure()
plt.hist(Ac_flow_out,np.arange(np.max(245),step=2),color='k')
plt.savefig('img/Ac_flow_out.pdf')

#%% set up unweighted adjacency graphs and compute # connected components / diameters

L_adj = L_g!=0
L_components = sp.sparse.csgraph.connected_components(L_adj,directed=False)

Ac_adj = Ac_g!=0
Ac_strong_components = sp.sparse.csgraph.connected_components(Ac_adj,directed=True,connection
    ='strong')
Ac_weak_components = sp.sparse.csgraph.connected_components(Ac_adj,directed=True,connection='
    weak')

Ac_weak_dists = sp.sparse.csgraph.dijkstra(Ac_adj+Ac_adj.T)
Ac_weak_diameter = np.max(Ac_weak_dists)

# construct directed adjacency graph of everything
C_g = L_g + Ac_g
C_adj = L_adj + Ac_adj
C_strong_components = sp.sparse.csgraph.connected_components(C_adj,directed=True,connection='
    strong')
C_weak_components = sp.sparse.csgraph.connected_components(C_adj,directed=True,connection='
    weak')

C_weak_dists = sp.sparse.csgraph.dijkstra(C_adj+C_adj.T)
C_weak_diameter = np.max(C_weak_dists)


#%% compute diameters of communication classes

L_diameters = np.zeros(L_components[0])
L_sizes = np.zeros(L_components[0])
for k in range(L_components[0]):
    rc = L_components[1]==k
    subgraph = L_adj[rc][:,rc]
    L_sizes[k] = len(subgraph)
    L_diameters[k] = np.max(sp.sparse.csgraph.dijkstra(subgraph))

Ac_diameters = np.zeros(Ac_strong_components[0])
Ac_sizes = np.zeros(Ac_strong_components[0])
for k in range(Ac_strong_components[0]):
    rc = Ac_strong_components[1]==k
    subgraph = Ac_adj[rc][:,rc]
    Ac_sizes[k] = len(subgraph)
    Ac_diameters[k] = np.max(sp.sparse.csgraph.dijkstra(subgraph))

C_diameters = np.zeros(Ac_strong_components[0])
C_sizes = np.zeros(Ac_strong_components[0])
for k in range(C_strong_components[0]):
    rc = C_strong_components[1]==k
    subgraph = C_adj[rc][:,rc]
    C_sizes[k] = len(subgraph)
    C_diameters[k] = np.max(sp.sparse.csgraph.dijkstra(subgraph))


#%%
def kura_rhs(t,theta,omega,n,K,A):
    coupling = np.zeros(n)
    for j in range(n):
        for i in range(n):
            coupling[j] += A[i,j]*np.sin(theta[i]-theta[j])

    return omega + (K/n)*coupling

#%%

T=100
t=np.linspace(0,T,2001);

n=len(L); # number of oscillators
#n=10

rad=np.ones((n,1));
thetai=2*np.pi*np.random.rand(n);
omega=np.random.rand(n)+0.5;

#A=np.random.rand(n,n);
#A[np.where(A<=0.5)] = 0;
#%%
Ks = [0,1,5,10,15,20,30,40,50,60,70,80,90,100]
ave_L_r = np.zeros(len(Ks))
ave_Ac_r = np.zeros(len(Ks))
ave_C_r = np.zeros(len(Ks))

for k,K in enumerate(Ks):
    print(k)

    # L
    sol = sp.integrate.solve_ivp(lambda t,theta:kura_rhs(t,theta,omega,n,K,L_adj),(0,T),
        thetai,t_eval=t)
    t = sol.t
    theta = sol.y

    r = np.abs(np.mean(np.exp(1j*theta),axis=0))
    ave_L_r[k] = np.mean(r[int(279/3):])

    plt.figure()
    plt.plot(t,theta.T%(2*np.pi),color='k',alpha=.01)
    plt.savefig('img/phase/L_'+str(K)+'.pdf')

    # Ac
    sol = sp.integrate.solve_ivp(lambda t,theta:kura_rhs(t,theta,omega,n,K,Ac_adj),(0,T),
        thetai,t_eval=t)
    t = sol.t
    theta = sol.y

    r = np.abs(np.mean(np.exp(1j*theta),axis=0))
    ave_Ac_r[k] = np.mean(r[int(279/3):])

    plt.figure()
    plt.plot(t,theta.T%(2*np.pi),color='k',alpha=.01)
    plt.savefig('img/phase/Ac_'+str(K)+'.pdf')


    # C
    sol = sp.integrate.solve_ivp(lambda t,theta:kura_rhs(t,theta,omega,n,K,C_adj),(0,T),
       thetai,t_eval=t)
    t = sol.t
    theta = sol.y

    r = np.abs(np.mean(np.exp(1j*theta),axis=0))
    ave_C_r[k] = np.mean(r[int(279/3):])

    plt.figure()
    plt.plot(t[::5],theta.T[::5]%(2*np.pi),color='k',alpha=.01)
    plt.savefig('img/phase/C_'+str(K)+'.pdf')


#%%

plt.figure()
plt.plot(Ks,ave_L_r,color='k',linestyle=':',marker='o')
plt.plot(Ks,ave_Ac_r,color='k',linestyle=':',marker='s')
plt.plot(Ks,ave_C_r,color='k',linestyle=':',marker='^')
plt.ylim([0,1])
plt.savefig('img/couple_strength.pdf')


#%%

plt.figure()
plt.plot(t,theta.T%(2*np.pi),color='k',alpha=.01)
#plt.ylabel('phase coherence')
#plt.xlabel('coupleing strength')
plt.savefig('img/phase_Ac_'+str(K)+'.pdf')

#%%
Ks = [1,10,20,30,40,50,75,100,150,200]
ave_Lg_r = np.zeros(len(Ks))
ave_Acg_r = np.zeros(len(Ks))

for k,K in enumerate(Ks):
    print(k)

    # L
    sol = sp.integrate.solve_ivp(lambda t,theta:kura_rhs(t,theta,omega,n,K,L_g),(0,T),thetai,
       t_eval=t)
    t = sol.t
    theta = sol.y

    r = np.abs(np.mean(np.exp(1j*theta),axis=0))
    ave_Lg_r[k] = np.mean(r[int(279/3):])

    plt.figure()
    plt.plot(t,theta.T%(2*np.pi),color='k',alpha=.01)
    plt.savefig('img/phase/Lg_'+str(K)+'.pdf')

    # Ac
    sol = sp.integrate.solve_ivp(lambda t,theta:kura_rhs(t,theta,omega,n,K,Ac_g),(0,T),thetai
        ,t_eval=t)
    t = sol.t
    theta = sol.y

    r = np.abs(np.mean(np.exp(1j*theta),axis=0))
    ave_Acg_r[k] = np.mean(r[int(279/3):])

    plt.figure()
    plt.plot(t,theta.T%(2*np.pi),color='k',alpha=.01)
    plt.savefig('img/phase/Acg_'+str(K)+'.pdf')


#%%

plt.figure()
plt.plot(Ks,ave_Lg_r,color='k',linestyle=':',marker='o')
plt.plot(Ks[:-1],ave_Acg_r[:-1],color='k',linestyle=':',marker='s')
plt.ylim([0,1])
#plt.ylabel('phase coherence')
#plt.xlabel('coupleing strength')
plt.savefig('img/couple_strength_g.pdf')
```