SPECTRA OF WEIGHTED GRAPH LAPLACIANS

TIM FERGUSON, IAN FORD, HIRAM GOLZE

1. Notation

Let G be an undirected, connected graph without multiple edges or graph loops, and let γ_{ij} represent the weight given to the edge connecting vertex i to vertex j. Define the graph Laplacian

$$L(G,\{\gamma\})_{ij} := \begin{cases} \gamma_{ij} & i \neq j, \\ -\sum_{k\neq i}\gamma_{ik} & i = j. \end{cases}$$

We will consider edge weights satisfying $\sum \gamma_{ij} = 1$ and are interested in investigating such edge weights which minimize the second largest eigenvalue of $L(G,\gamma)$. Two useful properties are that 0 is a simple eigenvalue of $L(G,\gamma)$ and the trace of $L(G,\gamma) = -2\sum\gamma_{ij} = -2$.

2. Gradient Decent And Code Implementation

The largest non-zero eigenvalue of the graph Laplacian controls the rate of convergence of the consensus algorithm. To find the optimum edge weights to ensure fastest convergence, we minimize this eigenvalue as a function of the edge weights. We do this by the method of gradient descent.

Perturbation theory tells us that the derivative of an eigenvalue λ with respect to an edge weight γ_{ij} is given by $< v, Mv > / < v, v >$ where v is the corresponding eigenvector and M is defined by

$$M_{uv} = -\delta_{uj}\delta_{uv} - \delta_{iv}\delta_{uv} + \delta_{ui}\delta_{vj} + \delta_{uj}\delta_{vi}$$

This gives us the components of the gradient of λ. To minimize the gradient subject to the constraint that $\sum\gamma_{ij} = 1$, we choose random starting edge edge weights and take a small step in the direction of $-\bigtriangledown(\lambda) + w$, where w is the vector who coordinates are all equal to the average of the coordinates of $\bigtriangledown(\lambda)$. This is implemented in code using networkx as follows:

```
def graph(nodes,edges):
    G = Graph()
    G.add_nodes_from(nodes)
    G.add_weighted_edges_from(edges)
    return G
```

This simplifies the process of building a graph as a networkx Graph object.

```
def pert(i,j,n): #perturbation corresponding to edge (i,j), n nodes
    M=zeros((n,n))
    M[i,j] = 1
    M[i,i] = -1
```

```
    M[j,i] = 1
    M[j,j] = -1
    return M
```

This defines the matrices M that we need to compute the gradient as above.

```
def eiglap(G): #returns the graph Laplacian and its spectrum
    L = -laplacian_matrix(G)
    lamda, v = eigh(L) #eigh automatically orders the eigenvalues
    return lamda, v, L
```

This returns the Laplacian of the inputted graph as well as information about its spectrum.

```
def grad(v,G):
    nodes=G.nodes()
    edges=G.edges()
    gradient = zeros(len(edges))
    for i in range(len(edges)):
        M=pert(edges[i][0],edges[i][1],len(nodes))
        gradient[i] = dot(v,dot(M,v))/dot(v,v)
    return gradient
```

This returns the gradient of the eigenvalue corresponding to the eigenvector v as described above.

```
def minimize(G,eps,N): #eps is the step size, N is the number of steps
    for i in range(N):

        edges = G.edges(data=True)
        #e.g. edges[0] might equal (0,1,{'weight'=0.25})

        edge_weights = zeros(len(edges))
        #just want to reserve the right amount of memory

        for j in range(len(edges)):
            edge_weights[j] = G[ edges[j][0] ][ edges[j][1] ][ 'weight' ]
            #creates an array of edge weights in the same order as the edges

        lamda, v, L = eiglap(G)

        gradient = grad(v[:,-2],G)
        #the second largest eigenvalue is the second from the end

        edge_weights -= eps*( gradient-sum(gradient)*ones(gradient.shape)/len(edges) )
        #takes a small step in the direction of most rapid decrease
        #subject to the constraint sum(edge_weights)=1

        for j in range(len(edges)):
            G[ edges[j][0] ][ edges[j][1] ][ 'weight' ]=edge_weights[j]
            #changes the weights to their new weights
```

```
return G
```

This is where gradient descent gets implemented.

3. Local Maximums And The Spectrum

We now consider some common graphs. Let K_n, C_n, and S_n denote the complete graph, cyclic graph, and star graph on n vertices respectively and B_n denote the complete binary tree with height n.

Conjecture 1. *Let $n \geq 2$ and G equal K_n, C_n, or S_n, then the second largest eigenvalue of $L(G)$ has a local maximum when the edge weights are equal. For $n \geq 1$ and G equals B_n, the second largest eigenvalue of $L(G)$ has a local maximum when all of the edge weights of the same depth are equal.*

We now state a proposition giving the structure of the spectrum of $L(G)$ with uniform edge weights for G equals K_n, C_n, or S_n.

Proposition 2. *Let $n \geq 2$ and the edge weights be uniform.*

(i) *The non-zero eigenvalues of $L(K_n)$ are all equal to $\frac{2}{1-n}$.*
(ii) *The jth eigenvalue of $L(C_n)$ is*

$$\frac{2}{n}\left(-1+\cos\left(\frac{2\pi j}{n}\right)\right).$$

(iii) *The non-zero eigenvalues of $L(S_n)$ are all equal to $\frac{1}{1-n}$ except for the smallest one which is equal to $\frac{n}{1-n}$.*

These cases lead us to the following conjecture.

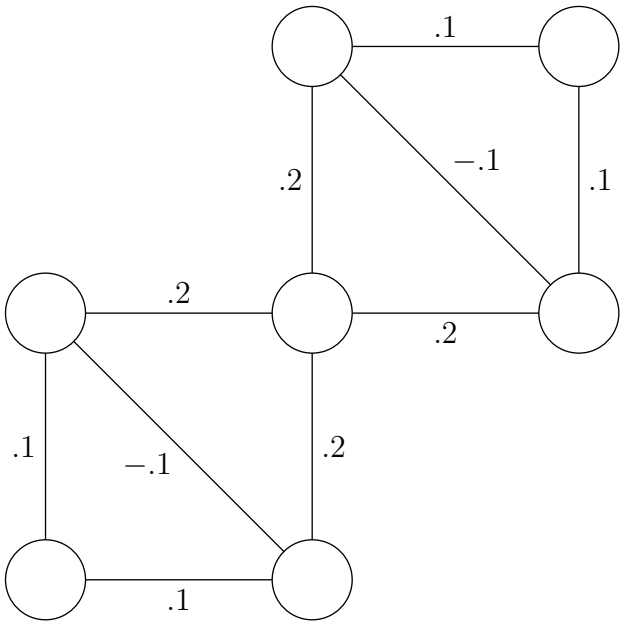
Conjecture 3. *The second largest eigenvalue of $L(G)$ is not simple.*

4. Negative Edge Weights

The only constraint that we have given on the edge weights is that they sum to one. This leaves the possibility open for some of the edge weights to be negative. One way to visualize what positive and negative edge weights represent is to imagine each vertex having a specified temperature. A positive edge weight between two vertices means that these two vertices will tend towards the same temperature. The larger the edge weight, the faster the rate at which they will tend towards the same temperature. A negative edge weight between two vertices means that these two vertices will to tend to different temperatures. The larger the edge weight, the faster the rate at which they will tend to different temperatures. Some edges are more important than others. For example, as seen below, the edges near a bottle neck are more important than ones that are farther away.

Here we give an example of a graph G such that the second largest eigenvalue of $L(G)$ has a local maximum when some edge weights are negative.

Example. Graph with negative edge weights


.1
.2
−.1
.1
.2
.2
.1
−.1
.2
.1