Readings

- Lecture notes on network exploration.
- Newman 6.14.3 on random walks.
- Newman 18.3.1 (network navigation).
- Lecture notes on agent-based modeling.

Problem 1. (2 points) Modularity is a Covariance

Consider a connected and aperiodic graph G with adjacency matrix $\mathbf{A}$. Suppose that G has two labeled clusters, which we'll label C_1 and C_2. The node membership vector $\mathbf{z}$ is defined so that $z_i = 1$ means that node i is in cluster 1. Let $\{X_t\}$ be a simple random walk on this graph. Define the sequence of random variables

$$S_t \triangleq \begin{cases} +1 & X_t \in C_1 \\ -1 & X_t \in C_2 \ . \end{cases} \tag{1}$$

That is, the sign of S_t tells us whether X_t is currently on a node in cluster 1 or a node in cluster 2. We assume that the initial distribution of X_0 is π, the stationary distribution. This implies that $\mathbb{P}(X_t = i) = \pi_i$ for all t.

You might imagine that, in a graph with two well-separated clusters, it's relatively unlikely for the random walk to move between them. That is, "most of the time," we should have $S_t = S_{t+1}$. One way to quantify this intuition is via the so-called *one-step auto-covariance*

$$\mathrm{Cov}(S_t, S_{t+1}) \triangleq \mathbb{E}[S_t S_{t+1}] - \mathbb{E}[S_t]\mathbb{E}[S_{t+1}] \ . \tag{2}$$

In this problem, you will show that this autocovariance is closely related to the modularity of the graph.

Part (a)

Compute $\mathbb{E}[S_t]$ and $\mathbb{E}[S_{t+1}]$ in terms of quantities like a_{ij}, k_i, and $\delta(z_i, z_j)$. The simplest way to approach this is by using the law of total expectation, conditioning on the event $X_t = i$ for each node i.

Part (b)

Compute $\mathbb{E}[S_t S_{t+1}]$. This is the trickiest part of the calculation! Here's my suggested first step, again using the law of total expectation:

$$\mathbb{E}[S_t S_{t+1}] = \sum_{i,j \in N} \mathbb{E}[S_t S_{t+1} | X_t = i, X_{t+1} = j] \mathbb{P}(X_{t+1} = j, X_t = i) \ . \tag{3}$$

Part (c)

Argue that

$$\mathrm{Cov}(S_t, S_{t+1}) = k_1 Q + k_2 \ , \tag{4}$$

for some constants k_1 and k_2, and determine the value of each constant. That is, another way to view the modularity is as a direct measure of the correlation between the cluster labels of a simple random walk.

Problem 2. (2 points)

Please use your software of choice to solve the following problems, making sure to show both commented code and clearly-labeled outputs. The only thing you are **not** allowed to do is use the Mesa framework for agent-based modeling in Python.

Part (a)

Write a function `random_walk(G, n_steps, i)` which simulates `n_steps` timesteps of a simple random walk on a graph `G`, starting at node `i`. This function should return a list or array containing the labels of the nodes visited by the walk. For example:

```
W = random_walk(G, n_steps, i)
W[t] # location of walk at time t
```

Part (b)

Access any small-ish connected graph (no more than 1,000 nodes), and use your function to simulate a random walk on this graph for at least 10^5 timesteps.

Part (c)

Create a scatterplot:

- Each point corresponds to a node i.
- The horizontal coordinate is π_i.
- The vertical coordinate is the fraction of time that the random walk spent on node i.

Also plot the line of equality $y = x$. Of course, we are expecting to find that the points on the scatterplot are close to the line of equality. Please make sure that your plot is carefully labeled.

Problem 3. (2 point)

Please use your software of choice to solve the following problems, making sure to show both commented code and clearly-labeled outputs. The only thing you are **not** allowed to do is use the Mesa framework for agent-based modeling in Python.

Note. This problem is intentionally similar to the last problem.

Part (a)

Write a function `pagerank_walk(G, n_steps, i, alpha)` which simulates `n_steps` timesteps of a PageRank (teleporting) random walk on a graph `G`, starting at node `i`. Note that the user is allowed to specify the teleportation parameter `alpha`. This function should return a list or array containing the labels of the nodes visited by the walk. For example:

```
W = pagerank_walk(G, n_steps, i)
W[t] # location of walk at time t
```

Part (b)

Access any small-ish connected graph (no more than 1,000 nodes), and use your function to simulate a PageRank random walk on this graph for at least 10^5 timesteps.

Part (c)

Note. This part is the biggest difference relative to Problem 2.

Write a function `pagerank_matrix(G,alpha)` which computes the transition matrix for the PageRank random walk, again with user-specified teleportation parameter. Compute the stationary distribution π_i by finding the leading eigenvector of this matrix.

An optimal solution will not use for-loops!

Part (d)

Create a scatterplot:

- Each point corresponds to a node i.
- The horizontal coordinate is π_i.
- The vertical coordinate is the fraction of time that the random walk spent on node i.

Also plot the line of equality $y = x$. Of course, we are expecting to find that the points on the scatterplot are close to the line of equality. Please make sure that your plot is carefully labeled.