STAT 302: Statistical Computing **Winter 2024**

Final Project Description

Yikun Zhang *February 20, 2023*

Acknowledgement: Parts of this project are modified from the midterm project of STAT 133 taught by Professor Deborah Nolan in Fall 2016 at UC Berkeley.

1 Introduction

Technological advancement has been changing our daily life. Nowadays, people rely heavily on their cell phones to receive messages and communicate with others around the world. All these cell phone communications are made possible by a concept called *ad hoc wireless network*[1], which has no centralized nodes or fixed structures. Instead, such a network structure is dynamically determined by the current locations and usages of all the communication devices in the nearby region. In other words, all the cell phones and other communication devices can create and join the ad-hoc wireless networks "on the fly".

A very basic aspect of ad hoc networks that people need to understand is how the communication and complete connectivity changes with respect to the broadcasting power of each node. Increasing the power level allows one to send messages over larger distances.

2 Project Descriptions

This project consists of three main tasks.

1. Start by writing a function to generate nodes in a given (two-dimensional) ad hoc network. The function should meet the following conditions:

   (a) The name of the functions is `genNodes()`.

   (b) It has one input argument: `n`, which is the number of nodes to generate.

   (c) The return value of `genNodes()` is an $n \times 2$ matrix with the first column representing the x coordinates and the second column as the y coordinates of the nodes.

   Use the *acceptance-rejection sampling* in Lecture 6 to randomly generate the nodes of the network according to a pre-specified node density function `nodeDensity()`. This node density function has been written in R and can be downloaded on our Canvas page. It takes two input arguments x and y, which should be two numeric vectors of the same length. The function returns a numeric vector of values that are proportional to the node density value at the (x, y) pairs. To read this function into the current R environment, run

   ```
   source("nodeDensity.R")
   ```

   in an R code chunk of your .Rmd file. The two-dimensipnal contour plot and three-dimensional perspective plot of the node density function are shown in Figure 1 and Figure 2 below.

   After writing the function `genNodes()`, you need to generate $n = 10000$ nodes from `genNodes()` and use a scatter plot or contour plot to visualize the generated nodes or node density. This visualization

[1] See https://en.wikipedia.org/wiki/Wireless_ad_hoc_network.

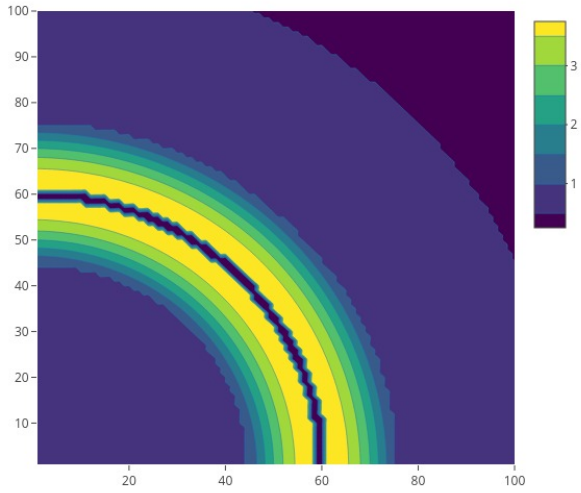

Figure 1: A two-dimensional contour plot of `nodeDensity()` in a region of interest. The contours are proportional to the density of nodes.

helps you confirm that the function `genNodes()` is indeed generating nodes according to the required density function `nodeDensity()`.

2. For a given configuration of nodes, we intend to find the smallest radius $R_c > 0$ such that the network is connected, *i.e.*, there is a path passing through nodes in the network between each pair of nodes. Please write an R function called `findRc()` to output R_c for a given collection of nodes. This function has the following properties:

   (a) The name of the function is called `findRc()`.

   (b) The first input argument is `nodes`, which is a two-column matrix of the x and y coordinates/locations of the nodes (*e.g.*, generated from the function `genNodes()` in Task 1).

   (c) The second input argument is `tol` with a default value of 0.05, which is the tolerance level for how close we need to get to the true value of R_c for the provided node configuration.

   (d) The return value is a numeric variable that holds the value of R_c (or a value that is close to it under the tolerance level `tol`).

   In this task, we are interested in finding the smallest power level R_c that leads to a connected network. Here, we assume that the power level is proportional to the radius $\bar{R}$ of a circle centered on the node, and any two nodes in the network are in direct communication (*i.e.*, connected by an edge in the network) if the distance between them is less than $\bar{R}$.

   The task of finding R_c relies on some probability arguments and a searching algorithm, which are described in Section 3. There, you will also be asked to write a couple of helper functions for `findRc()`.

3. Select a range of sample sizes for n (*e.g.*, $n = 50, 100, 150, 200, 250$, respectively). Then, for each n, generate 1000 networks via the function `genNodes()` in Task 1 and find the value of R_c for each of these 1000 networks via the function `findRc()` in Task 2. Examine the distribution of these R_c values for each n. There are some questions that you need to answer in words with some supporting plots:

   (a) How does R_c, the smallest radius such that the network is connected, change with different node configurations?

   (b) Explore the distribution of R_c (for each n). Is it symmetric, skewed, heavy-tailed, and multimodal?

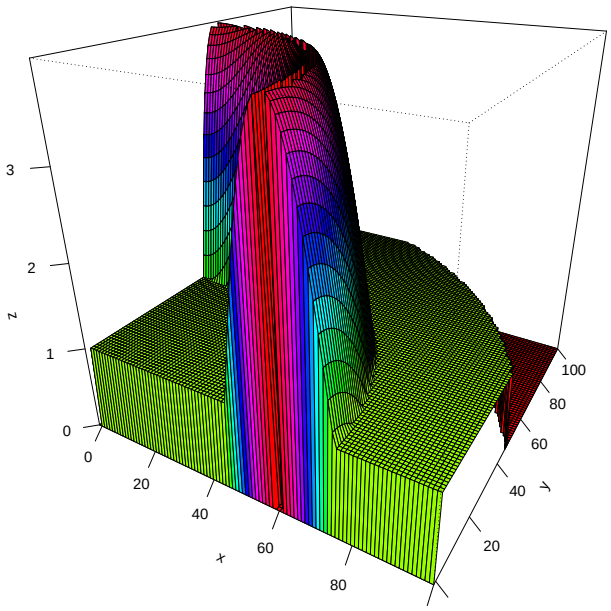

Figure 2: A three-dimensional perspective plot of `nodeDensity()` in a region of interest. Note that there are river curves passing through the center of the region so that no nodes are located near the river.

(c) For $n = 50$, plot the network of connected nodes as well as their associated edges for four of your 1000 n-node configurations corresponding *roughly* to the minimum, mean, median, and maximum values of R_c.

Note that in order to plot a network, you should keep track of the n-node configurations that correspond to the minimum, mean, median, and maximum values of R_c. For example, if you keep track of the seed value at the start of each simulation, then, when you find that the m^{th} simulation corresponds to the median value for R_c, you can set the seed to the m^{th} seed that you have saved and recreate the nodes. In addition, you should connect those pairs of nodes whose distances are less than R_c when plotting the network.

3 Mathematical Background

A set of nodes is said to be connected if there is a path passing through nodes between each pair of nodes. For our wireless ad hoc network context, when the power level of each device reaches a certain level, the communication between any two devices in the network can be achieved by connecting along other devices that are within the broadcasting distance of each other. On the contrary, if the power level is too low, then there may be some nodes that are unable to connect to other nodes, or the nodes in the network may form two or more disjoint connected subsets.

3.1 Pairwise Distance Matrix

Suppose that our ad hoc network has n nodes in total. We define a distance matrix $D \in \mathbb{R}^{n \times n}$, whose (i, j)-entry D_{ij} stores the (Euclidean) distance between nodes i and j with $1 \leq i, j \leq n$. This matrix D is symmetric and has all zeros on its diagonal, which can be computed via the function `dist()` in `R`; see also Section 4.

3.2 Random Walk Between the Nodes

In order to determine if a network/graph is connected, one possible avenue is to study the transition matrix $P \in \mathbb{R}^{n\times n}$ of a random walk between nodes in the network. Specifically, for a particular power level $\bar{R} > 0$, we consider a simplified case where a node has only three neighbors within $\bar{R}$. Then, in our random walk, a message located at this node will hop at random to one of these three nodes or will stay at its current node, and these four possible scenarios are equally likely. Following this rule, a message will hop around nodes in the network in a random fashion.

The transition matrix $P \in \mathbb{R}^{n\times n}$ is defined to characterize the probabilities of a random walk between nodes in a network from its current state to the next state. In particular, $P_{ij} = 0$ if the distance from node i to node j is larger than $\bar{R}$. Otherwise, let $\mathcal{S}_i$ be the index set of nodes whose distances to node i are less than or equal to $\bar{R}$, and $P_{ik} = \frac{1}{|\mathcal{S}_i|}$ for each $k \in \mathcal{S}_i$, where $|\mathcal{S}_i|$ is the total number of elements (*i.e.*, node indices) in $\mathcal{S}_i$. There are several properties of the transition matrix P (notice that it depends on $\bar{R}$):

1. Note that $i \in \mathcal{S}_i$ for all $i = 1, ..., n$. That is, any node in the network is connected to itself, so the diagonal entries of P are in $(0, 1]$.
2. The sum of each row of P is always equal to 1, *i.e.*, $\sum_{k\in\mathcal{S}_i} P_{ik} = \sum_{k\in\mathcal{S}_i} \frac{1}{|\mathcal{S}_i|} = 1$ or $P\mathbf{1} = \mathbf{1}$ with $\mathbf{1} \in \mathbb{R}^n$ being a vector with all one entries.
3. If $\boldsymbol{v}_m \in \mathbb{R}^n$ is a vector of probabilities that a message is at any one of the n nodes at the current state m, then $\boldsymbol{v}_{m+1} = P^T\boldsymbol{v}_m$ is the distribution of locations of the message at the next state $m + 1$. And $\boldsymbol{v}_{m+2} = P^T\boldsymbol{v}_{m+1}$ is the distribution at the state $m + 2$. (We can also think of $\boldsymbol{v}_m$ as the distribution of many messages across the network.)
   - The reason why we use the transpose of the transition matrix P is that the i^{th} column of P (or the i^{th} row of P^T) characterizes the probabilities of a message moving from all the nodes to node i.
4. The distribution of locations of the message will eventually settle down, *i.e.*, there exists some $\boldsymbol{v} \in \mathbb{R}^n$ such that
$$P^T\boldsymbol{v} = \boldsymbol{v}. \tag{1}$$
   The probability vector $\boldsymbol{v} \in \mathbb{R}^n$ is also known as the *stationary distribution* of P or the random walk that it characterizes.
   - The equation (1) indicates that the stationary distribution $\boldsymbol{v}$ is the eigenvector of the transition matrix P^T associated with the eigenvalue 1.
   - All the eigenvalues of P^T (or equivalently, P) have magnitude less than or equal to 1.
   - If the network of nodes is connected, then there exists a unique stationary distribution $\boldsymbol{v}$. In this case, the largest eigenvalue of P^T is real and has value 1. All the other eigenvalues have their magnitudes less than 1.

▶ The above properties imply that the size of the *second largest* eigenvalue of P is the key to determine if a network is completely connected or not.

Write a helper function called `findTranMat()` to find the transition matrix based on a distance matrix D and a value of $\bar{R}$:

(a) The first input argument is `mat`, which is the distance matrix $D \in \mathbb{R}^{n\times n}$ that stores all the pairwise distances between nodes in the network.

(b) The second input argument is `R`, which is the value of the power level $\bar{R}$. Any pair of nodes whose distance is less than or equal to $\bar{R}$ will be considered being connected.

(c) The function returns the transition matrix $P \in \mathbb{R}^{n \times n}$ based on the above two inputs.

Write another helper function called `getEigen2()`, which returns the magnitude of the second largest eigenvalue of a matrix:

(a) There is only one input argument of this function as `mat`, which is a $n \times n$ transition matrix P.

(b) The function returns a numeric value, which is the magnitude of the second largest eigenvalue of the input matrix P.

3.3 Range of R_c

What is the smallest possible value that R_c can be?

- Since each node must be connected to at least one other node in the network, R_c must be at least as large as the greatest row-wise minimum of the distance matrix D, *i.e.*,

$$R_c \geq \max\{a_1, a_2, ..., a_n\} \quad \text{with} \quad a_i = \min\{D_{i1}, D_{i2}, ..., D_{in}\} \quad \text{for} \quad i = 1, ..., n.$$

What is the largest possible value that R_c can take?

- If R_c is greater than the maximum distance in a row, then the corresponding node will be connected to all of the other nodes. Thus, we know that R_c should be no greater than the smallest row-wise maximum, *i.e.*,

$$R_c \leq \min\{b_1, b_2, ..., b_n\} \quad \text{with} \quad b_i = \max\{D_{i1}, D_{i2}, ..., D_{in}\} \quad \text{for} \quad i = 1, ..., n.$$

Write a helper function called `findRange()`, which finds the range of possible values to search for R_c based on the above observations:

(a) There is only one input argument of this function as `mat`, which is the distance matrix $D \in \mathbb{R}^{n \times n}$.

(b) The function returns a numeric vector of length 2, which are the minimum and maximum values to search for R_c.

3.4 Bisection Method: Searching for R_c

Our function `findRange()` gives us an interval in which we can search for R_c. One key fact is that if a network is connected for a particular value $\bar{R}^*$, then it is connected for all larger values $\bar{R} > \bar{R}^*$. We want to find the smallest $\bar{R}$ (within a certain tolerance level) that gives us a connected network.

One approach is to follow the idea of the bisection method[2] that we learn in Lecture 8 [Zhang, 2023]:

1. Denote the endpoints of the interval returned from `findRange()` by $\bar{R}_{\min}$ and $\bar{R}_{\max}$, respectively.
2. Compute the middle point as $\bar{R}_0 = \frac{\bar{R}_{\min} + \bar{R}_{\max}}{2}$.
3. If $\bar{R}_0$ gives a completely connected network, then we assign $\bar{R}_{\max} \leftarrow \bar{R}_0$. Otherwise, we assign $\bar{R}_{\min} \leftarrow \bar{R}_0$.
4. Repeat Steps 2 and 3 until $|\bar{R}_{\min} - \bar{R}_{\max}|$ is less than the tolerance level `tol`.

The above procedures will be the main part of our function `findRc()` described in Section 2.

[2] See https://en.wikipedia.org/wiki/Bisection_method.

4 Useful Resources

Below are some functions that you may find useful:

- `set.seed()` and `.Random.seed` – used to set and save the seed for the random number generator.
- `dist()` – computes the pairwise distances between rows in a matrix or data frame.
- `eigen()` – computes eigenvalues and eigenvectors of a matrix. Be sure to read about all the arguments in its documentation.
- `eigs()` in the `RSpectra` package – calculate the largest k eigenvalues (you may also omit the computations of eigenvectors through `opts`).
- `expand.grid()` – a useful function when creating a grid from vectors, if you need to search over a grid.
- `Mod()` – calculates the magnitude of complex numbers.

Sanity checks for some of the functions that you have been asked to write are available as `testsForAdHocProject.R` and `adhocCheckPublish.R` on our Canvas page as well. We also provide a `PracticeW.Random.seed.Rmd` file for an example of how to use `.Random.seed` to keep track of and reset the random number generator.

5 Submission Requirements

- You can work in a group with size ≤ 3 for this final project.
- Please submit BOTH the knitted PDF and the original .Rmd file on Gradescope. In your submitted files, write down the names of all your group members.
- *Only one member of your group needs to make the submission on Gradescope.* There is a group submission function on Gradescope. If multiple members in your group have made the submission, we will randomly grade one of the writeups and assign the grades to all students in this group.
- Everyone should make more or less contributions to the project if you are working in a group. *Highlight at the beginning of your .Rmd and knitted PDF files the contributions of every group member to the project.* If you work on the project by yourself, you can skip this part.

References

Y. Zhang. STAT 302 Lecture Slides, 2023. URL https://zhangyk8.github.io/teaching/stat302_uw.