Stat 133, Fall 05
Mini-Project: Ad Hoc Network Simulation
Due: Monday, 31 Oct

Wireless networks are all around us. Cell phones communicate with a base-station to send and receive calls. Calls are relayed from base-station to base-station as the cell phone moves away from one and closer to another. A new idea of organizing networks is to avoid the need for a central base-station that coordinates communications. Instead, messages are relayed by "hopping" from one node to the next to the next until it reaches its destination. In other words, one can send a message by using other devices in the network to relay the message to the next device, and so on. These are called *ad hoc* networks because there is no centralized node or fixed structure or topology for the network. Instead, devices can move over time, and dynamically enter and exit the network. And so the route a message takes from one device to another depends on the other nodes.

Ad hoc networks are very promising and becoming important. At their most immediately practical, ad hoc networks can allow nodes outside of a regular network to communicate by piggy-backing off of nodes within the network. Think of driving along and being between base-stations and so your cell phone call would be dropped. But because of ad hoc networks, your data is relayed through cell phones in other cars closer to the base stations. More ambitiously, ad hoc networks might be used in controlling traffic on highways by allowing cars communicate with each other.

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. Increased power levels allow one to send a message over a larger distance.

- Start by generating a basic ad hoc network. Place 75 nodes at random on the 2-dimensional grid shown in Figure 1, where the node density is proportional to the function shown in Figure 2. Note the area of interest includes a "river" where the density of nodes along the river is higher, except for locations in the river and along the banks of the river. This density function is supplied as an R function called **nodeDensity**. A vector of x and a vector of y coordinates are taken as inputs to the function, and the return value is a vector of values that are proportional to node density at the (x, y) pairs.

- Search over different power levels and determine whether the nodes are connected or not for the various power levels. Remember that for the set of nodes to be connected means that it is possible for any two nodes to communicate by having a message travel along nodes that are within broadcasting distance of each other. If the power level is very high then all nodes will be able to broadcast directly to each other. On the other hand, if the power level is too low then a node may not be able to connect to any other node, or the nodes may form two disjoint connected subsets. We are interested in finding the smallest power level that leads to a connected network. We assume here that power level is proportional to the radius, R, of a circle centered on the node, and two nodes are in direct communication if the distance between them is less than R.

- How does the value of R_c, the smallest radius such that the network is connected, change with different node configurations? Repeat the above process of finding R_c for 1000 random node configurations generated according to the node density in Figure 2. Explore the

distribution of R_c. Is it symmetric, long tailed, similar to any of the "standard" distributions (normal, log-normal, exponential, gamma, etc.) Plot the network of connected points for four of your 1000 node configurations corresponding roughly to the min, median, mean, and maximum values of R_c.

Background A probability model for messages moving on the network can help us determine if the network is completely connected. For a particular power level R, suppose a node has 3 neighbors within R of it. Then 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. The message hops around on the network in a random fashion, and we can describe its random "walk" via a transition matrix for a Markov Chain. That is, for a message located at node i, $i = 1, \ldots, N$ the chance it moves to node j, $j = 1, \ldots, N$, is 0 if these two nodes are further than R away from each other, or it is $1/k_i$ where k_i is the number of nodes within R of node i (including node i).

So, for example, if v_m is the $N \times 1$ vector of probabilities that a message is at any one of the N nodes at one "instant", then $Pv_m = v_{m+1}$ is the distribution of locations of the message at the next instant, and $Pv_{m+1} = v_{m+2}$ for the next instant, where P represents the transition probabilities described in the previous paragraph. (We can also think of v_m as the distribution of many messages across the network.)

Mathematical properties of transition matrices tell us that the distribution of the locations of the message settles down, which implies that there is some v where $Pv = v$. We see from this equation that the steady state is the eigenvector of the transition matrix associated with the eigenvalue of 1. Further, the eigenvalues of P are all real and less than or equal to one, and if the network of nodes is fully connected, there is one unique steady-state solution. In this case, only the largest eigenvalue is one, which implies that the size of the second largest eigenvalue is key to determining if the network is connected.

Assignment You are to turn in your code, the *random seed* used in your simulation (saved in a .rda file – the seed must be unique to you), a description of the distribution of R_c including one plot, and four plots of connected networks for the four values of R_c (min, max, median, and mean). Functions that you may find helpful for this homework:

- **.Random.seed** - used to set and save the seed for the random number generator.
- **runif** - a pseudo-random number generator that takes vector arguments.
- **eigen** - computes eigenvalues and eigenvectors. Be sure to read about all of the arguments.
- **which** - can be very handy for figuring out which elements in a vector have a certain property.
- **expand.grid** - a very useful function when searching over a grid.
- **contour** - makes contour plots (used for the figures sown here). Be careful with the arguments, z is a matrix corresponding to a grid of (x, y) values.
- **dist** - computes all pairwise distances between rows in a matrix or data frame.

You may want to use either the treebeard or radagast machines.

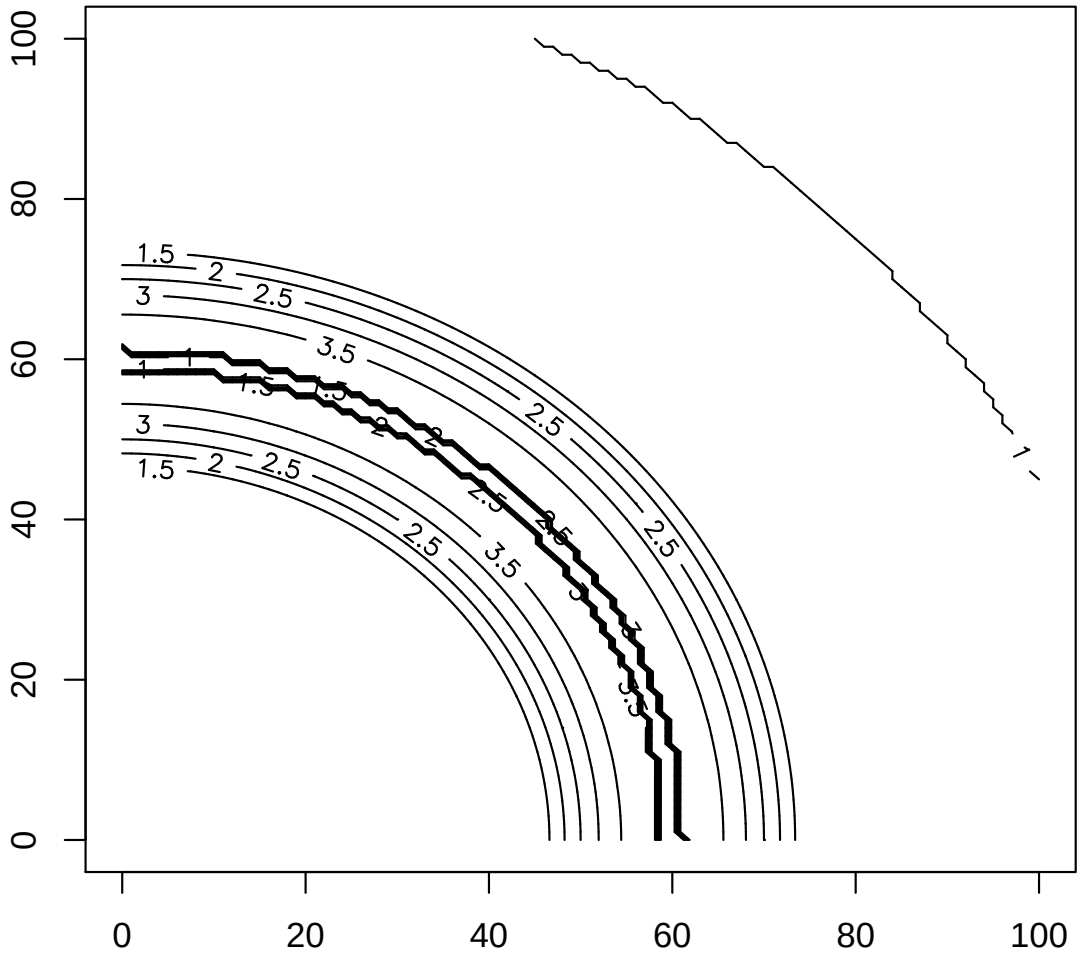

Figure 1: Contour plot of the region of interest. The contours are proportional to the density of nodes.

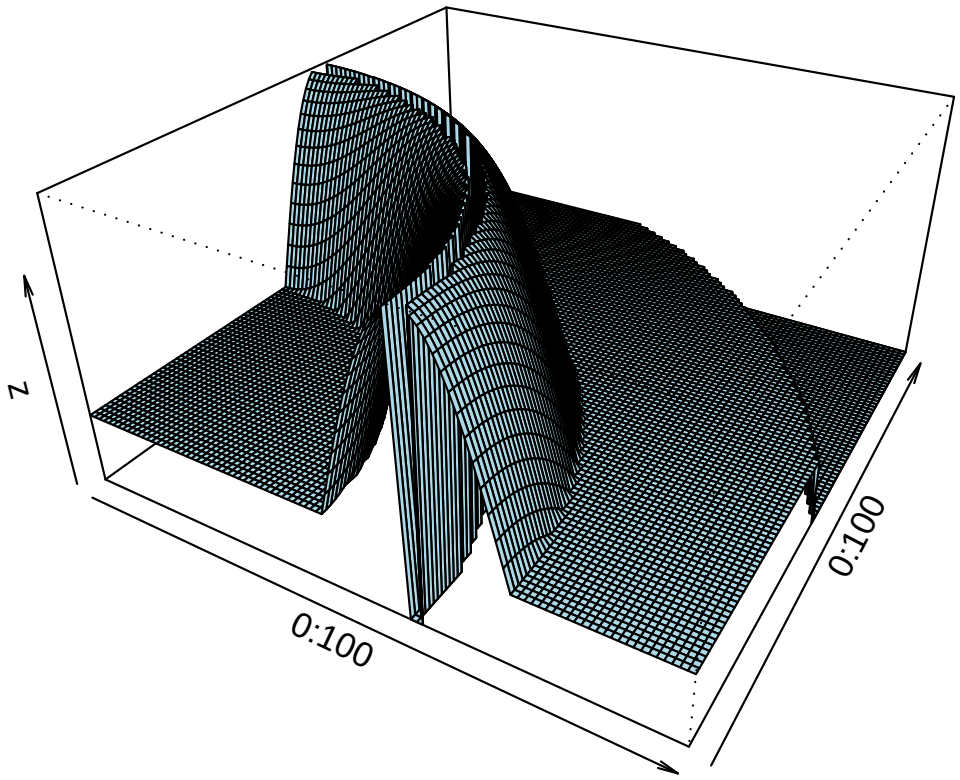

Figure 2: A three-dimensional perspective plot of the region of interest. Note that the river curves through the center of the region and no nodes are located near the river.