Name : Dae Woong Ham (No Group Worked Alone) Group Number 40 on BCourses SID: 26222439

Task 1: Finding the maximum

```
source("http://www.stat.berkeley.edu/users/nolan/data/nodeDensity.R")

x = seq(0, 100, length.out = 200)
y = seq(0, 100, length.out = 200)
g = expand.grid(x, y)
g$density = nodeDensity(g$Var1, g$Var2)
max(g$density)
```

```
## [1] 3.983466
```

This first code chunk gave me a maximum of around 3.98 which will be used in my next code chunk for the basis of the acceptance rejection sampling.

Task 1: Creating a function to generate the coordinates

```
genNodes = function(n) {
  nodes_coordinates = matrix(0, nrow = n, ncol = 2, dimnames = list(1:n, c("x", "y")))
  x = runif(3*n, min = 0, max = 100)
  y = runif(3*n, min = 0, max = 100)
  density = nodeDensity(x, y)
  z = runif(3*n, min = 0, max = 3.98 + 0.01)
  interested_subset = z < density
  x = x[interested_subset]
  y = y[interested_subset]
  if (length(x) >= n) {
    nodes_coordinates[ , 1] = x[1:n]
    nodes_coordinates[ , 2] = y[1:n]
    return(nodes_coordinates)
  }
    while(length(x) < n) {
      a = runif(n, min = 0, max = 100)
      b = runif(n, min = 0, max = 100)
      density = nodeDensity(a, b)
      c = runif(n, min = 0, max = 3.98 + 0.01)
      subset = c < density
      a = a[subset]
      b = b[subset]
      x = c(x, a)
      y = c(y, b)
    }
    nodes_coordinates[ , 1] = x[1:n]
    nodes_coordinates[ , 2] = y[1:n]
    return(nodes_coordinates)
}
```

This code chunk helps get a randomely generated x and y coordinates by using acceptance rejection sampling. I chose 3*n number to initially sample and if insufficient chose to use a while loop to continue to sample. Obviously if it was too much I just chose to take the first n x and y coordinates. I fundamentally built this code on subsetting and concatenatning values to my x and y coordinates.

Task 2: Creating Transition Matrix helper function

```
findTranMat = function(mat, R) {
  TranMat = (mat <= R) + 0
  k = apply(TranMat, 1, sum)
  for (i in 1:nrow(mat)) {
    TranMat[i, ][TranMat[i, ] == 1] = 1/(k[i])
  }
  return(TranMat)
}
```

This will be the Transition Matrix that was built on first using logicals to change my distance matrix to a matrix of 0's and 1's. The reason why I did this is because the 1's represent the information that there is a possibility for a certain configuration to be within the radius. The 1's were then summed by the apply function to find that "k" value I need later to take the fraction of, and finally used a for loop to populate my matrix with these counts.

Task2: Creating eigenvalue helper function

```
require(RSpectra)
```

```
## Loading required package: RSpectra
```

```
getEigen2 = function(mat) {
  return(eigs(mat, 2, which = "LM", opts = list(retvec = FALSE))[[1]][2])
}
```

This function is my getEigenvalue function that I used the package RSpectra for a faster numerical way to get my second eigenvalue. This will later help for efficiency.

Task2: Creating my range helper function

```
findRange = function(mat) {
  zero_removed_mat = matrix(mat[!mat == 0], nrow = nrow(mat), ncol = nrow(mat) - 1, byro
w = TRUE)
  min_range = max(apply(zero_removed_mat, 1, min))
  max_range = min(apply(zero_removed_mat, 1, max))
  return(c(min_range, max_range))
}
```

This last helper function findRange will help me find the range of my Rc values. I first got rid of all my zeros in my distance matrix and then took the minimum and maximum of it in that respective order to find my range.

Task 2: Creating the desired function to find Rc.

```
findRc = function(nodes, tol = 0.05) {
  dist_matrix = as.matrix(dist(nodes))
  Rc = mean(findRange(dist_matrix))
  R1 = findRange(dist_matrix)[1]
  R2 = findRange(dist_matrix)[2]
  if (1 - Mod(getEigen2(findTranMat(dist_matrix, Rc))) < tol) #make it bigger
    Rc = c(mean(c(Rc, R2)), R2)
  else #make it smaller
    Rc = c(R1, mean(c(Rc, R1)))
  while(abs(Rc[2] - Rc[1]) >= tol) {
    Rc = c(Rc, mean(Rc))
    if (1 - Mod(getEigen2(findTranMat(dist_matrix, Rc[3]))) < tol) {
      Rc = Rc[!Rc == min(Rc)]
    }
    else
      Rc = Rc[!Rc == max(Rc)]
  }
  return(max(Rc))
}
```

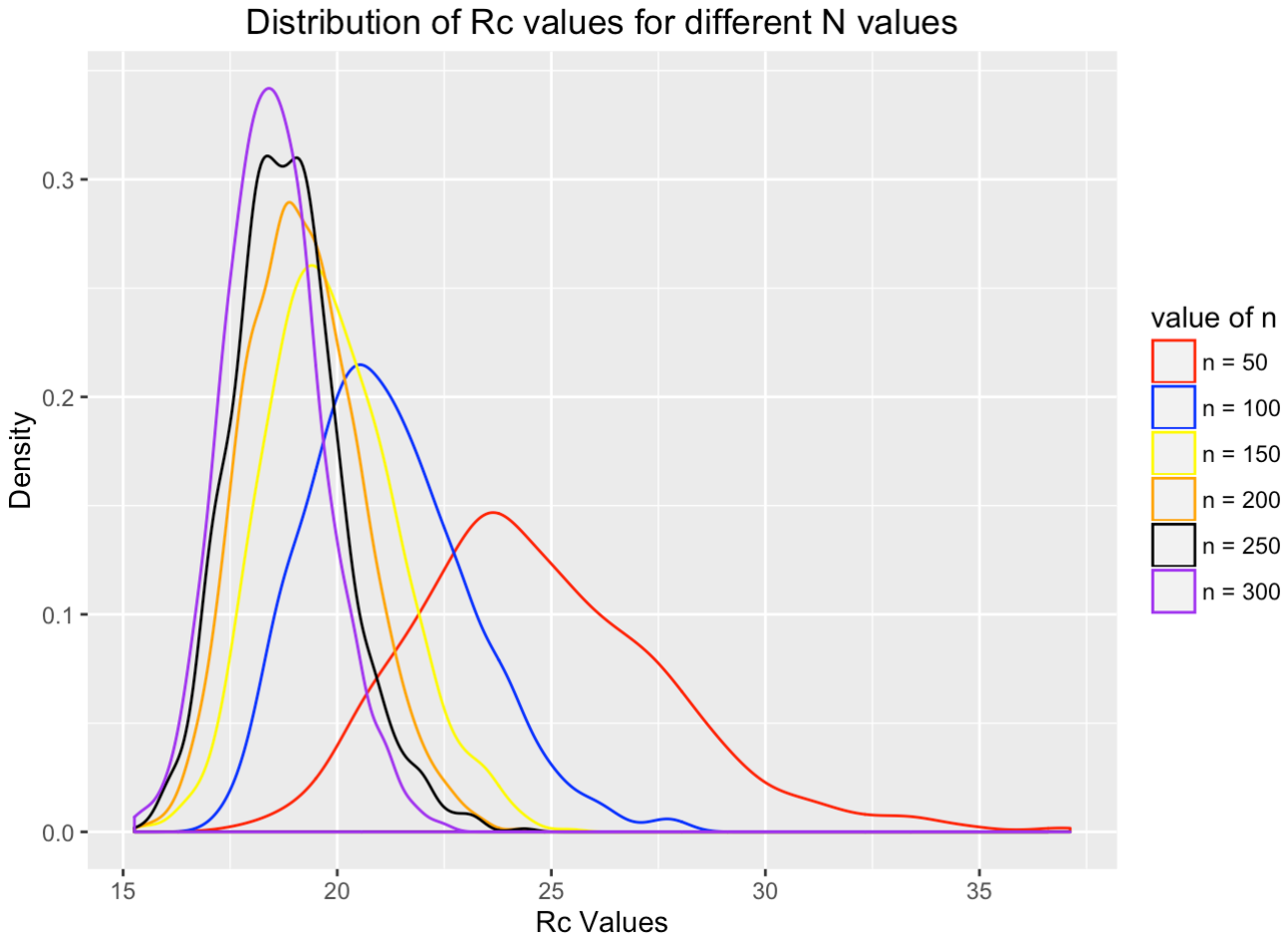
This is the key function to this project that uses all my helper functions to generate the minimum Rc value to have a connected network. My logic was to first zero in on the right value by keep taking the middle of my findRange function and going left or right accordingly by using an if and else check. I first got my initial 2 values I was interested then I saved it to my variable. Then the while loop was there for basically keep doing this halving process until I have my correct Rc value to the right tolerance level. Note my method was based on creating a vector of length 3 and removing the min or maximum of that and keep averaging these values once it's length 2. Then I repeat this process. Also, I never had to use absolute value function for my "if statement" because the eigenvalue never exceeds one so it is sufficient to do 1 subtract the second eigenvalue and check if it's within the tolerance level.

Task3: Creating different n-node Rc distributions

```
require(ggplot2)
```

```
## Loading required package: ggplot2
```

```
n = c(50, 100, 150, 200, 250, 300)

Rc_values_50 = rep(0, 1000)
nodes_50 = replicate(1000, genNodes(n[1]))
for (i in 1:1000) Rc_values_50[i] = findRc(nodes_50[, , i])

Rc_values_100 = rep(0, 1000)
nodes_100 = replicate(1000, genNodes(n[2]))
for (i in 1:1000) Rc_values_100[i] = findRc(nodes_100[, , i])

Rc_values_150 = rep(0, 1000)
nodes_150 = replicate(1000, genNodes(n[3]))
for (i in 1:1000) Rc_values_150[i] = findRc(nodes_150[, , i])

Rc_values_200 = rep(0, 1000)
nodes_200 = replicate(1000, genNodes(n[4]))
for (i in 1:1000) Rc_values_200[i] = findRc(nodes_200[, , i])

Rc_values_250 = rep(0, 1000)
nodes_250 = replicate(1000, genNodes(n[5]))
for (i in 1:1000) Rc_values_250[i] = findRc(nodes_250[, , i])

Rc_values_300 = rep(0, 1000)
nodes_300 = replicate(1000, genNodes(n[6]))
for (i in 1:1000) Rc_values_300[i] = findRc(nodes_300[, , i])

values_of_n = rep(n, each = 1000)
n_vs_Rc = data.frame(Rc = c(Rc_values_50, Rc_values_100, Rc_values_150, Rc_values_200, R
c_values_250, Rc_values_300), n = values_of_n)

ggplot(n_vs_Rc) + geom_density(aes(Rc, color = factor(n))) + scale_color_manual(name =
"value of n", values = c("red", "blue", "yellow", "orange", "black", "purple"), labels =
 c("n = 50", "n = 100", "n = 150", "n = 200", "n = 250", "n = 300")) + labs(x = "Rc Valu
es", y = "Density", title = "Distribution of Rc values for different N values")
```



Here I created and stored 1000 random Rc values of different n node configuration that ranges from 50 to 300 in increments of 50. (length 6) I then chose to superpose the density plot on a single graph by creating a dataframe of the Rc value corresponding to each n value, hence 6000 rows. This data frame also allowed me to factor by my value of n showing 6 density curves allowing me to analyze the differences in the distribution. I also created a legend to represent the approriate labels with each color.

Analysis: First, all 6 density curves similarly show a right skewed plot with one peak all around 17.5-25 range. However, as the value of n grew larger and larger the peak became higher and higher and the peak (also the median in this case) also moved to a smaller Rc value or to the left. This definitly makes sense because the median of many greater node configurations should have a smaller Rc value to make it fully connected. Hence we see this pattern as values of n increase.

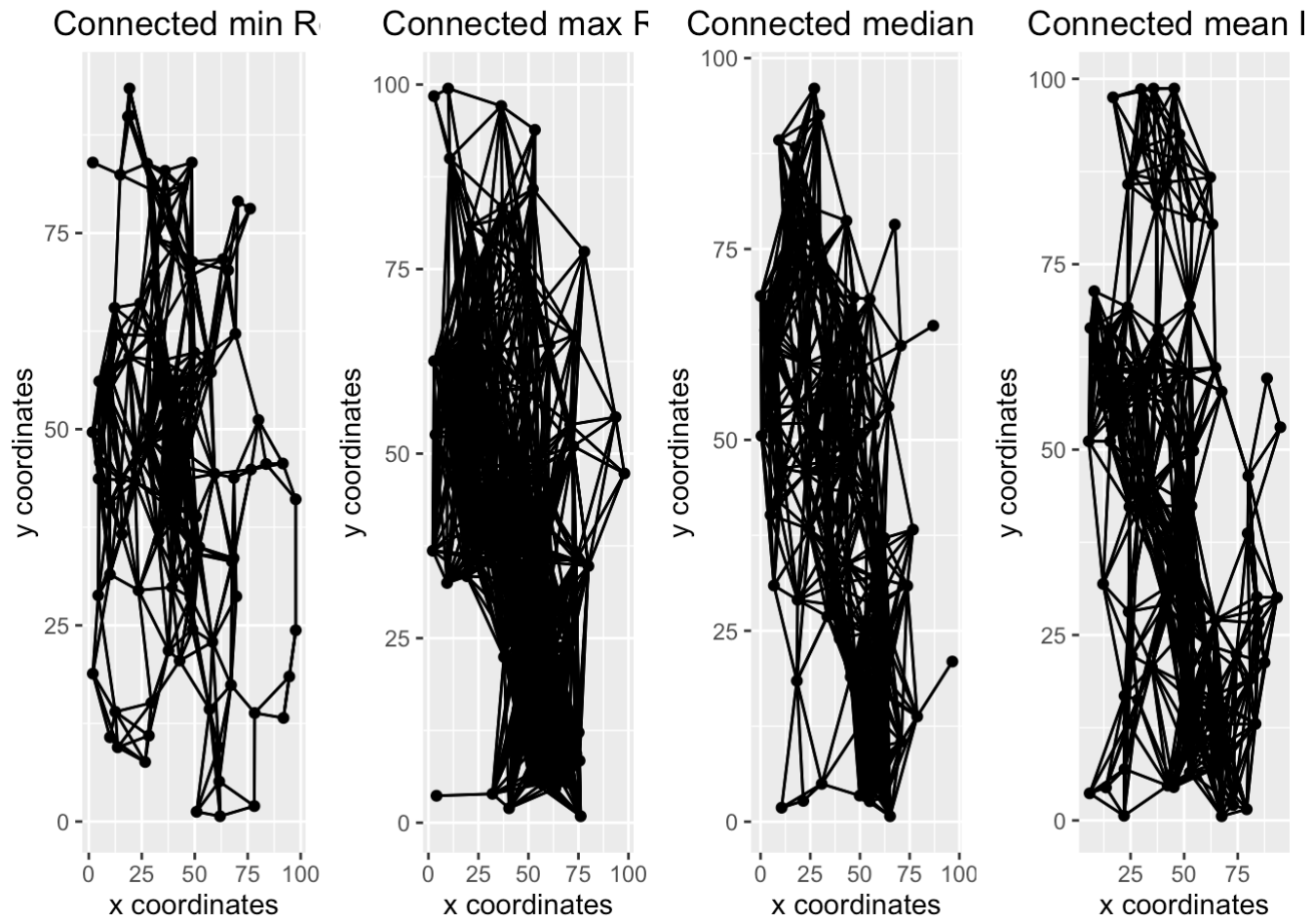
Task 3: Choosing n = 100 and plotting the connected network

```
minimum_element = which(Rc_values_100 == min(Rc_values_100))
min_nodes = nodes_100[, , minimum_element]
dist_matrix = as.matrix(dist(min_nodes))
TranMat = (dist_matrix <= Rc_values_100[minimum_element]) + 0
k = apply(TranMat, 1, sum)
x_start = rep(min_nodes[, 1], k)
y_start = rep(min_nodes[, 2], k)
x_end = vector(mode = "numeric", length = 0)
y_end = vector(mode = "numeric", length = 0)
for (i in 1:nrow(TranMat)) {
  for (j in 1:ncol(TranMat)) {
    if (TranMat[i, j] == 1) {
      x_end = c(x_end, min_nodes[j, 1])
      y_end = c(y_end, min_nodes[j, 2])
    }
  }
}
min_nodes = as.data.frame(min_nodes)
start_end_nodes = data.frame(x_start = x_start, x_end = x_end, y_start = y_start, y_end
= y_end)
min_network = ggplot(min_nodes, aes(x, y)) + geom_point() +
  geom_segment(data = start_end_nodes, aes(x = x_start, y = y_start, xend = x_end, yend
= y_end)) + labs(x = "x coordinates", y = "y coordinates", title = "Connected min Rc")

max_element = which(Rc_values_100 == max(Rc_values_100))
max_nodes = nodes_100[, , max_element]
dist_matrix = as.matrix(dist(max_nodes))
TranMat = (dist_matrix <= Rc_values_100[max_element]) + 0
k = apply(TranMat, 1, sum)
x_start = rep(max_nodes[, 1], k)
y_start = rep(max_nodes[, 2], k)
x_end = vector(mode = "numeric", length = 0)
y_end = vector(mode = "numeric", length = 0)
for (i in 1:nrow(TranMat)) {
  for (j in 1:ncol(TranMat)) {
    if (TranMat[i, j] == 1) {
      x_end = c(x_end, max_nodes[j, 1])
      y_end = c(y_end, max_nodes[j, 2])
    }
  }
}
max_nodes = as.data.frame(max_nodes)
start_end_nodes = data.frame(x_start = x_start, x_end = x_end, y_start = y_start, y_end
= y_end)
max_network = ggplot(max_nodes, aes(x, y)) + geom_point() +
  geom_segment(data = start_end_nodes, aes(x = x_start, y = y_start, xend = x_end, yend
= y_end)) + labs(x = "x coordinates", y = "y coordinates", title = "Connected max Rc")

median_element = which(abs(Rc_values_100 - median(Rc_values_100)) < 0.01)[1]
median_nodes = nodes_100[, , median_element]
dist_matrix = as.matrix(dist(median_nodes))
TranMat = (dist_matrix <= Rc_values_100[median_element]) + 0
k = apply(TranMat, 1, sum)
```

```
x_start = rep(median_nodes[, 1], k)
y_start = rep(median_nodes[, 2], k)
x_end = vector(mode = "numeric", length = 0)
y_end = vector(mode = "numeric", length = 0)
for (i in 1:nrow(TranMat)) {
  for (j in 1:ncol(TranMat)) {
    if (TranMat[i, j] == 1) {
      x_end = c(x_end, median_nodes[j, 1])
      y_end = c(y_end, median_nodes[j, 2])
    }
  }
}
median_nodes = as.data.frame(median_nodes)
start_end_nodes = data.frame(x_start = x_start, x_end = x_end, y_start = y_start, y_end
= y_end)
median_network = ggplot(median_nodes, aes(x, y)) + geom_point() +
  geom_segment(data = start_end_nodes, aes(x = x_start, y = y_start, xend = x_end, yend
= y_end)) + labs(x = "x coordinates", y = "y coordinates", title = "Connected median Rc"
)

mean_element = which(abs(Rc_values_100 - mean(Rc_values_100)) < 0.01)[1]
mean_nodes = nodes_100[, , mean_element]
dist_matrix = as.matrix(dist(mean_nodes))
TranMat = (dist_matrix <= Rc_values_100[mean_element]) + 0
k = apply(TranMat, 1, sum)
x_start = rep(mean_nodes[, 1], k)
y_start = rep(mean_nodes[, 2], k)
x_end = vector(mode = "numeric", length = 0)
y_end = vector(mode = "numeric", length = 0)
for (i in 1:nrow(TranMat)) {
  for (j in 1:ncol(TranMat)) {
    if (TranMat[i, j] == 1) {
      x_end = c(x_end, mean_nodes[j, 1])
      y_end = c(y_end, mean_nodes[j, 2])
    }
  }
}
mean_nodes = as.data.frame(mean_nodes)
start_end_nodes = data.frame(x_start = x_start, x_end = x_end, y_start = y_start, y_end
= y_end)
mean_network = ggplot(mean_nodes, aes(x, y)) + geom_point() +
  geom_segment(data = start_end_nodes, aes(x = x_start, y = y_start, xend = x_end, yend
= y_end)) + labs(x = "x coordinates", y = "y coordinates", title = "Connected mean Rc")

require(gridExtra)
```

```
## Loading required package: gridExtra
```

```
grid.arrange(min_network, max_network, median_network, mean_network, ncol = 4)
```



Here I chose the n = 100 network which I used the previous saved variables from my previous code chunk. Note I didn't have to use random seeding in any way because I saved all the 1000 possible node configurations in a variable. Then I could call it back by simple subsetting. Despite this really long code chunk, most of it is just a repeat of the basic structure for the 4 different summary statistic. To get the x and y starting and ending coordinate points I used the transition matrix and changed it into a logical matrix of 0 and 1 where I am counting the 1's like an indicator to check wheter I create "edges" between two nodes or not. In order to use the geom_segment to connect the lines properly, I needed to find 4 things: xstart, ystart, xend, yend. Hence I needed a dataframe to represent this and that's exactly what I did by first finding the xstart and ystart with simple replications, then xend and yend by actually double for looping through my transition matrix and getting the certain element of my saved coordinate points to return when checking if it's 1 in the logically converted transition matrix. Finally using gridExtra package I created side by side plots of the connected network of each summary statistics.

Analysis: What I observed is that the maximum Rc value connected node configuartion has one outlier or several clumped up outliers of nodes to bring the Rc value up extremely high hence making everything that is grouped up outside the outlier to have basically all lines going through each other. That's why the lines in the max summary statistics is extremely messy and highly dense in the clumped up area and has only one line connecting that outlier. The minimum is quite the opposite in terms of how many edges were produced. The minimum Rc value means that the randomely generated nodes were generally spread out and that's exactly how it's represented. In terms of an engineering standpoint, this is probably the most favorable graph because the minimum Rc value to get a fully connected network means the engineer can produce a rather compartively weaker power and still get a connected network because the nodes are all quite nicely distributed. The maximum on the other hand is something the engineer doesn't want because he would have to create a much more powerful connection to get all the nodes connected just because of one "bad" node that is an outlier. The median and mean in this case was

quite similar and hence produced similarly even graphs in the sense that there weren't a lot of edges created nor very little. This is probably the graph that the engineer wants to look at most to get the right power level that will represent the general distribution of the random node configurations most of the time.