Project1_505851728_005627440_505297814

April 18, 2022

1 Project1: Random Graphs and Random Walks

- 505851728 Yang-Shan Chen
- 005627440 Chih-En Lin
- 505297814 Rikako Hatoya

[1]:
```
install.packages("igraph")
install.packages("Matrix")
install.packages("pracma")
install.packages("resample")
install.packages("textTinyR")
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

also installing the dependencies ‘Rcpp’, ‘RcppArmadillo’, ‘BH’
```

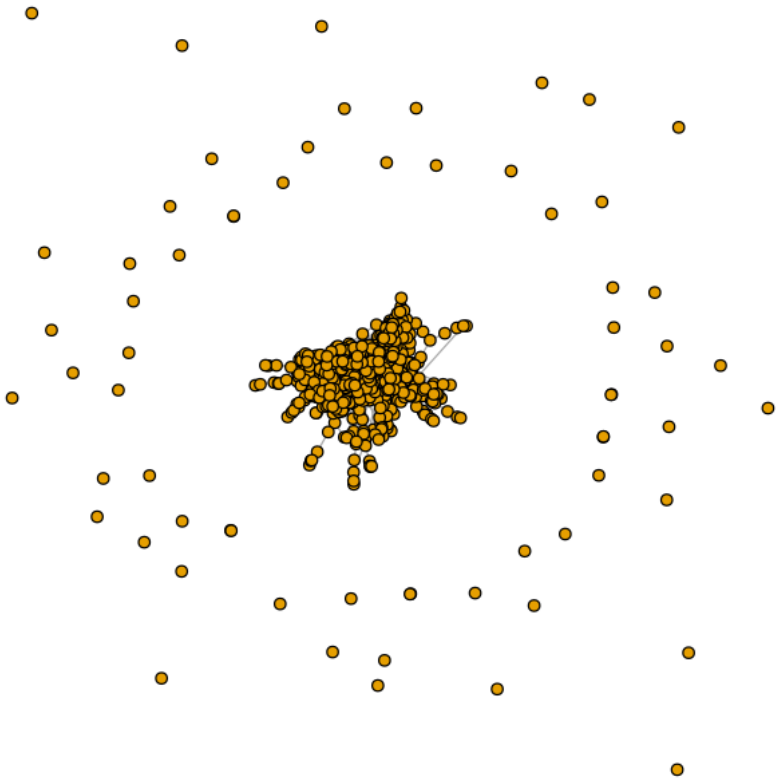
[5]:
```
library('igraph')
library('Matrix')
library('pracma')
library('resample')
library('textTinyR')
```

```
Attaching package: ‘igraph’


The following objects are masked from ‘package:stats’:

    decompose, spectrum


The following object is masked from ‘package:base’:

    union


Attaching package: ‘pracma’


The following objects are masked from ‘package:Matrix’:

    expm, lu, tril, triu
```

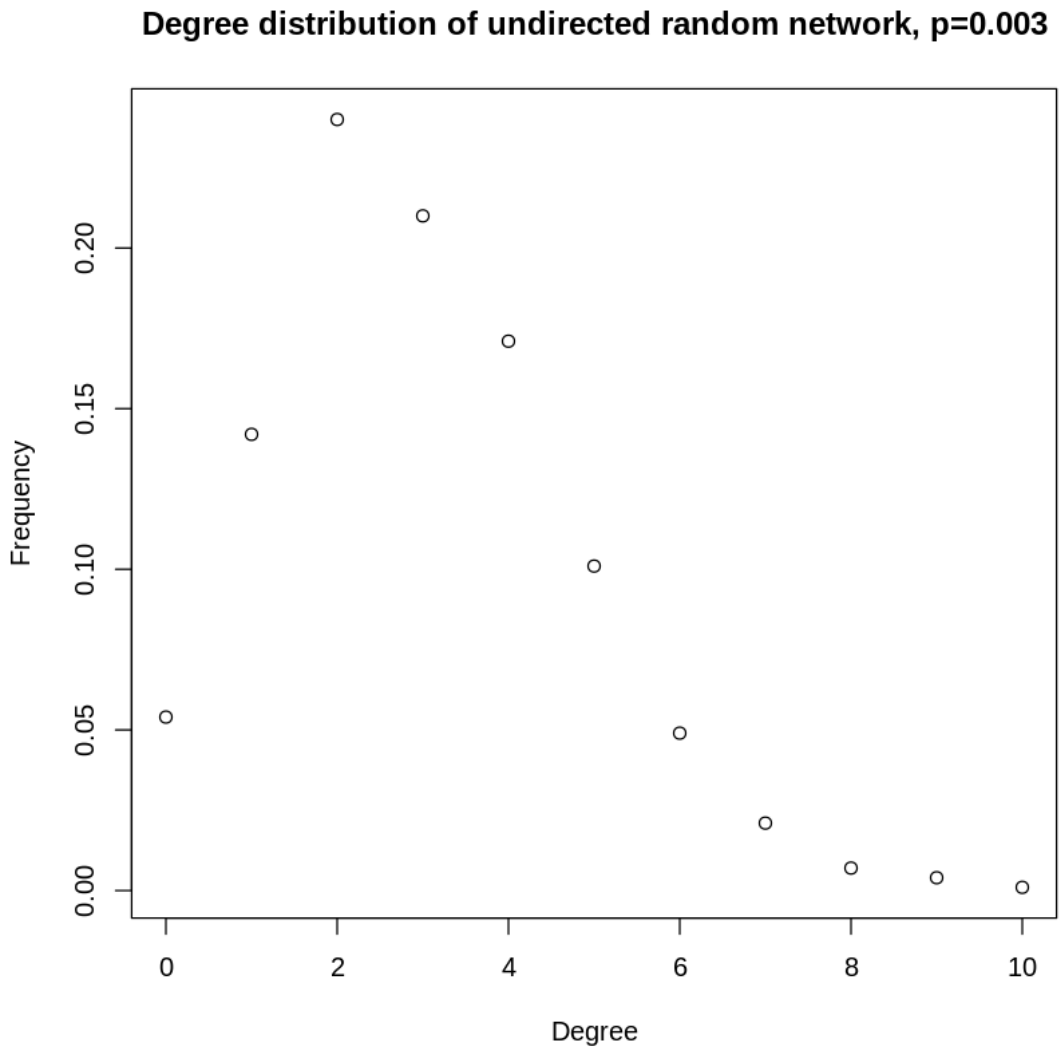
2 Part1

3 1(a)

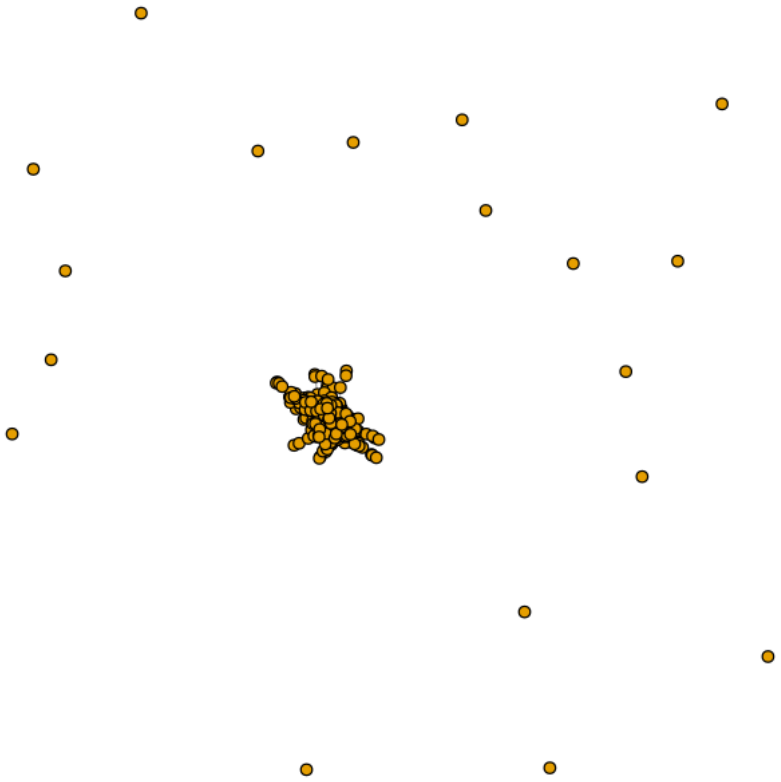
[ ]:
```r
g1 <- sample_gnp(1000, 0.003, directed = FALSE, loops = FALSE)
plot(g1, vertex.size=3, vertex.label=NA)
plot(seq_along(degree.distribution(g1)) - 1, degree.
 ↪distribution(g1),main="Degree distribution of undirected random network, p=0.
 ↪003",xlab="Degree",ylab="Frequency")
print(sprintf("Mean: %f",mean(degree(g1))))
print(sprintf("Variance: %f", var(degree(g1))))
print(sprintf("Theoretical Mean: %f", 1000*0.003))
print(sprintf("Theoretical Variance: %f", 1000*0.003*(1-0.003)))
```



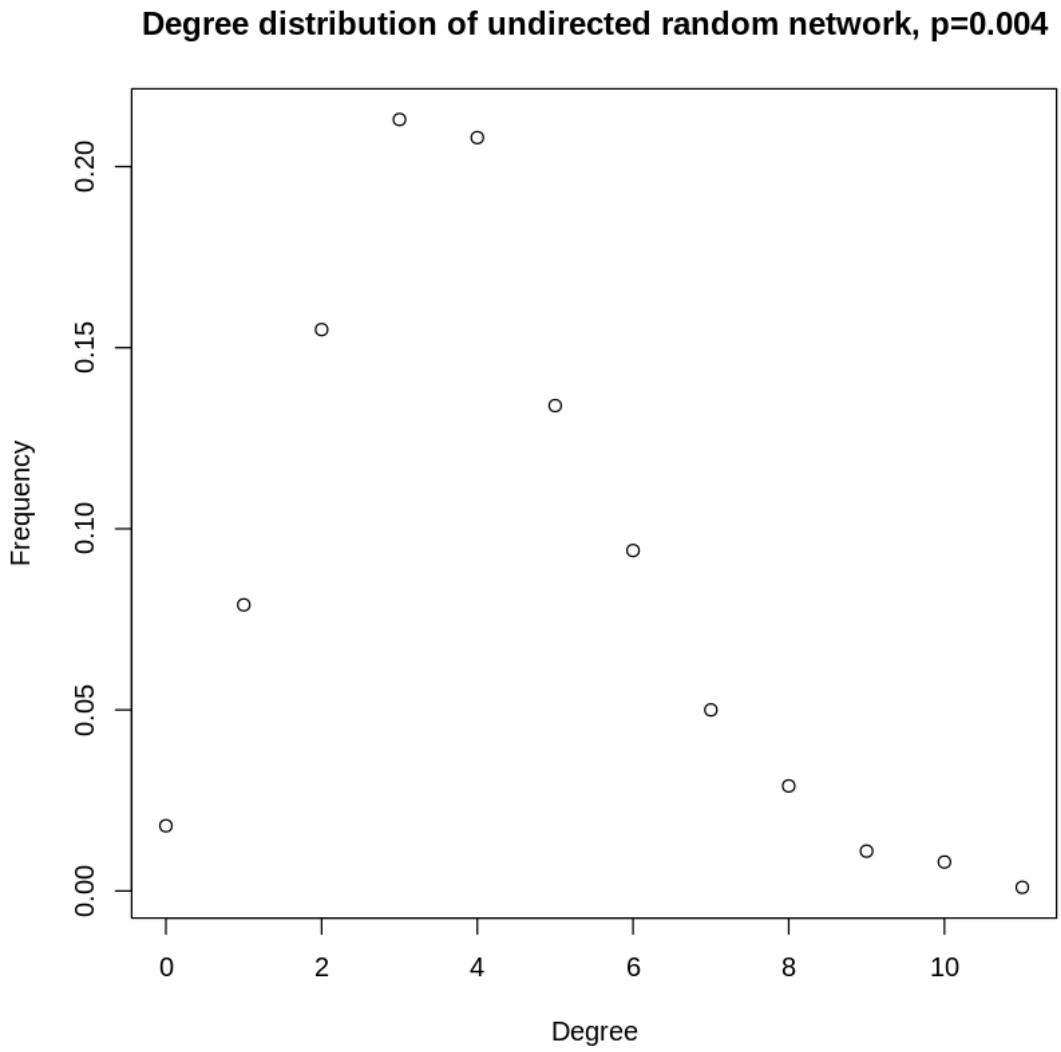
```
[1] "Mean: 2.984000"
[1] "Variance: 3.016761"
[1] "Theoretical Mean: 3.000000"
[1] "Theoretical Variance: 2.991000"
```



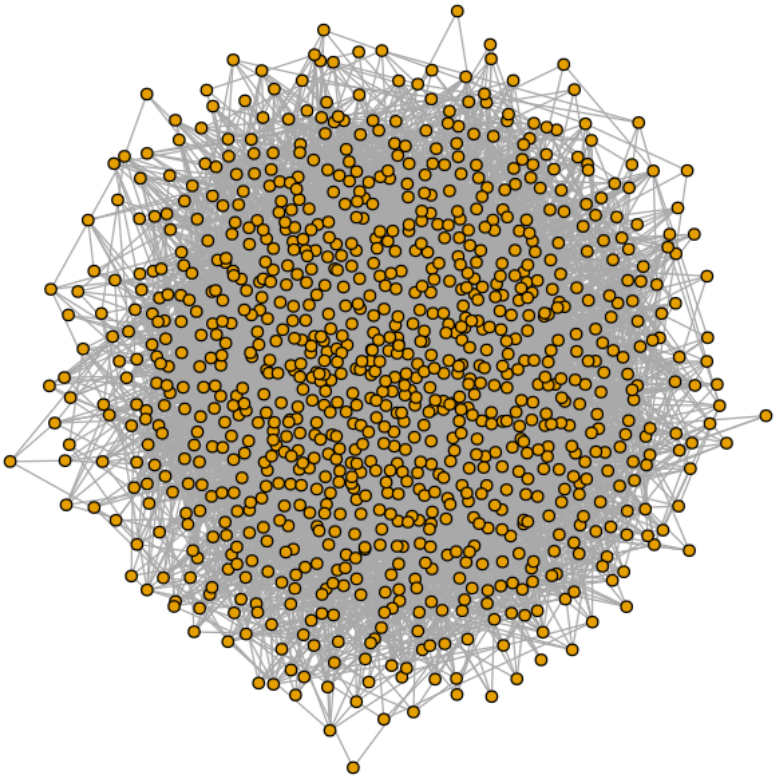
[ ]:
```
g2 <- sample_gnp(1000, 0.004, directed = FALSE, loops = FALSE)
plot(g2, vertex.size=3, vertex.label=NA)
plot(seq_along(degree.distribution(g2)) - 1, degree.
  ↪distribution(g2),main="Degree distribution of undirected random network, p=0.
  ↪004",xlab="Degree",ylab="Frequency")
print(sprintf("Mean: %f",mean(degree(g2))))
print(sprintf("Variance: %f", var(degree(g2))))
print(sprintf("Theoretical Mean: %f", 1000*0.004))
print(sprintf("Theoretical Variance: %f", 1000*0.004*(1-0.004)))
```



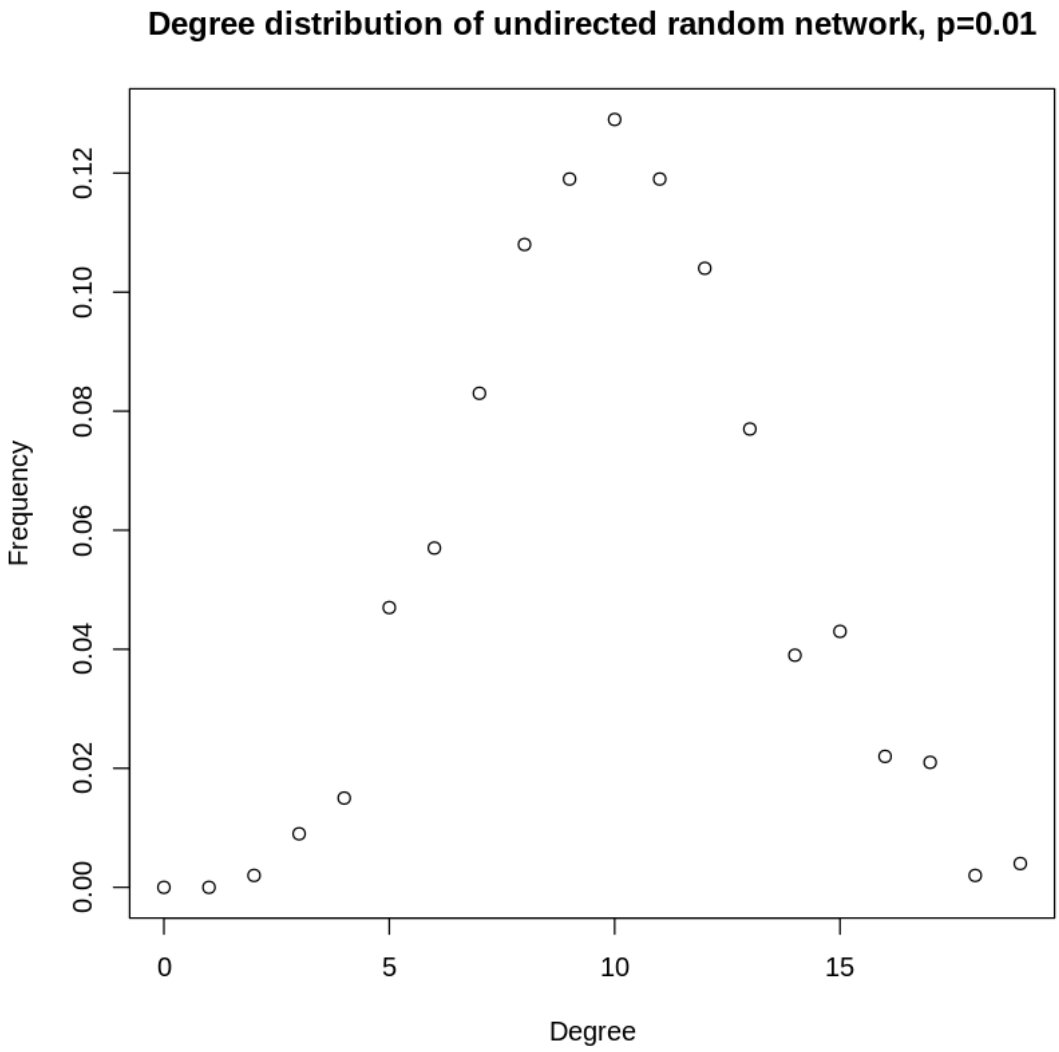
```
[1] "Mean: 3.866000"
[1] "Variance: 3.853898"
[1] "Theoretical Mean: 4.000000"
[1] "Theoretical Variance: 3.984000"
```



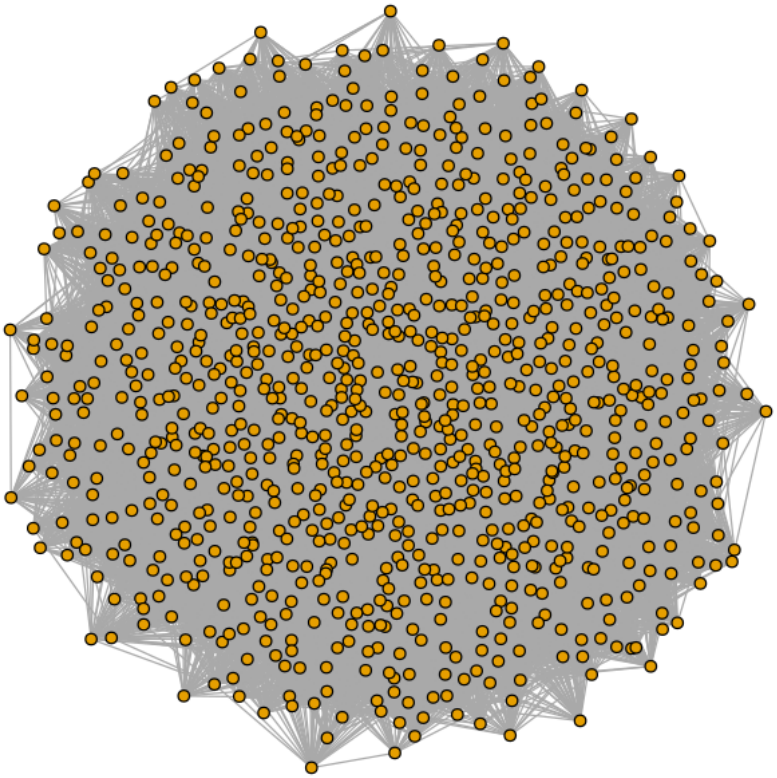
[ ]:
```r
g3 <- sample_gnp(1000, 0.01, directed = FALSE, loops = FALSE)
plot(g3, vertex.size=3, vertex.label=NA)
plot(seq_along(degree.distribution(g3)) - 1, degree.
  ↪distribution(g3),main="Degree distribution of undirected random network, p=0.
  ↪01",xlab="Degree",ylab="Frequency")
print(sprintf("Mean: %f",mean(degree(g3))))
print(sprintf("Variance: %f", var(degree(g3))))
print(sprintf("Theoretical Mean: %f", 1000*0.01))
print(sprintf("Theoretical Variance: %f", 1000*0.01*(1-0.01)))
```



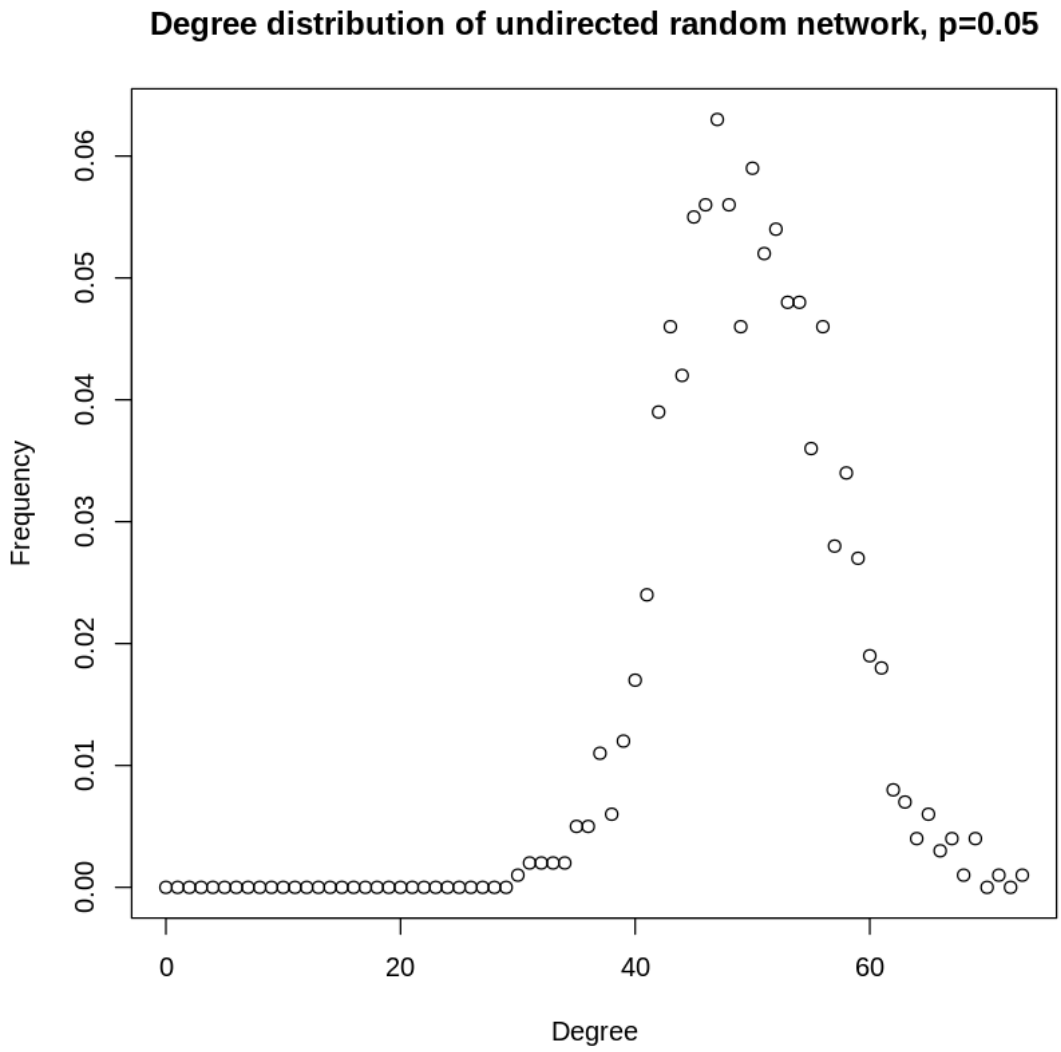
```
[1] "Mean: 10.044000"
[1] "Variance: 9.701766"
[1] "Theoretical Mean: 10.000000"
[1] "Theoretical Variance: 9.900000"
```



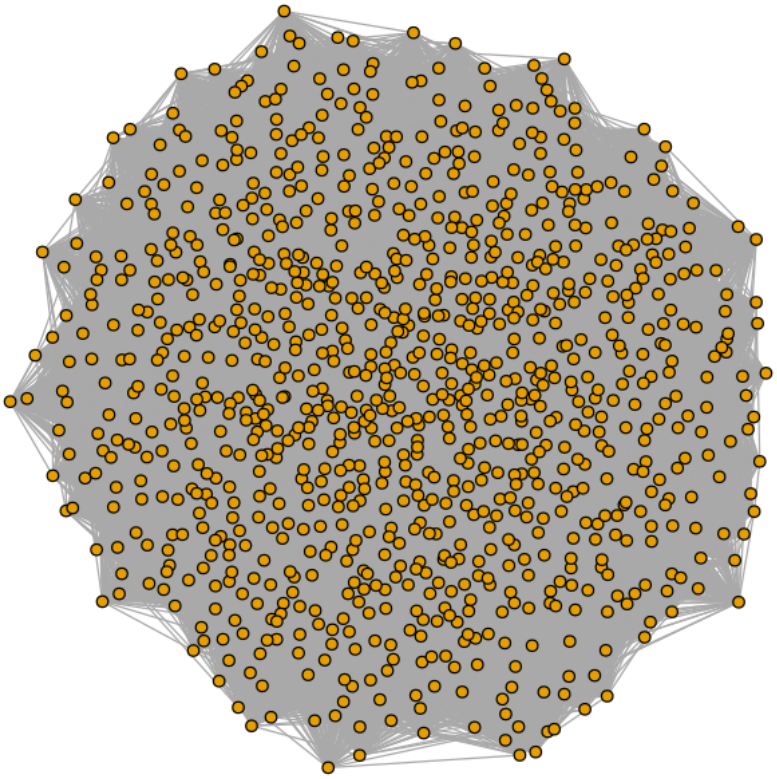
[ ]:
```
g4 <- sample_gnp(1000, 0.05, directed = FALSE, loops = FALSE)
plot(g4, vertex.size=3, vertex.label=NA)
plot(seq_along(degree.distribution(g4)) - 1, degree.
  ↪distribution(g4),main="Degree distribution of undirected random network, p=0.
  ↪05",xlab="Degree",ylab="Frequency")
print(sprintf("Mean: %f",mean(degree(g4))))
print(sprintf("Variance: %f", var(degree(g4))))
print(sprintf("Theoretical Mean: %f", 1000*0.05))
print(sprintf("Theoretical Variance: %f", 1000*0.05*(1-0.05)))
```



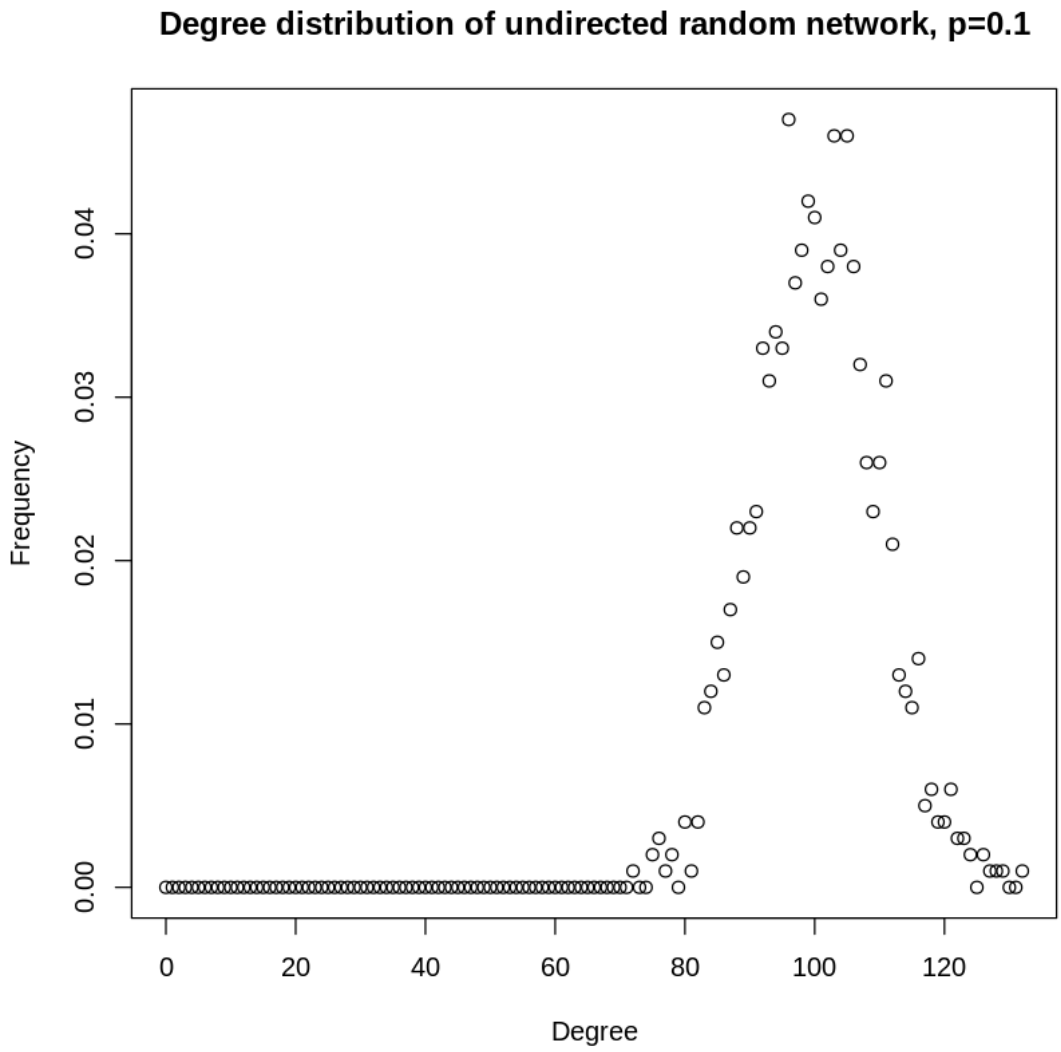
```
[1] "Mean: 49.868000"
[1] "Variance: 45.149726"
[1] "Theoretical Mean: 50.000000"
[1] "Theoretical Variance: 47.500000"
```



[ ]:
```
g5 <- sample_gnp(1000, 0.1, directed = FALSE, loops = FALSE)
plot(g5, vertex.size=3, vertex.label=NA)
plot(seq_along(degree.distribution(g5)) - 1, degree.
  ↪distribution(g5),main="Degree distribution of undirected random network, p=0.
  ↪1",xlab="Degree",ylab="Frequency")
print(sprintf("Mean: %f",mean(degree(g5))))
print(sprintf("Variance: %f", var(degree(g5))))
print(sprintf("Theoretical Mean: %f", 1000*0.1))
print(sprintf("Theoretical Variance: %f", 1000*0.1*(1-0.1)))
```



```
[1] "Mean: 100.344000"
[1] "Variance: 86.846511"
[1] "Theoretical Mean: 100.000000"
[1] "Theoretical Variance: 90.000000"
```



Binomial distribution is observed because given a graph with n vertices, the degree of each vertex is the sum of n-1 independent random variables and therefore, the degree distribution can be approximated as $P(k) \approx \binom{n}{k} p^k (1-p)^{n-k}$. The mean of the distribution is then $< k >= \sum_{k=0}^{n} k p_k = np$. Since $< k^2 >= \sum_{k=0}^{n} k^2 p_k = np(1-p) + p^2 N^2$, the variance is $ _{k}^{2} = (<k^{2}>-{2})=np(1-p)$.

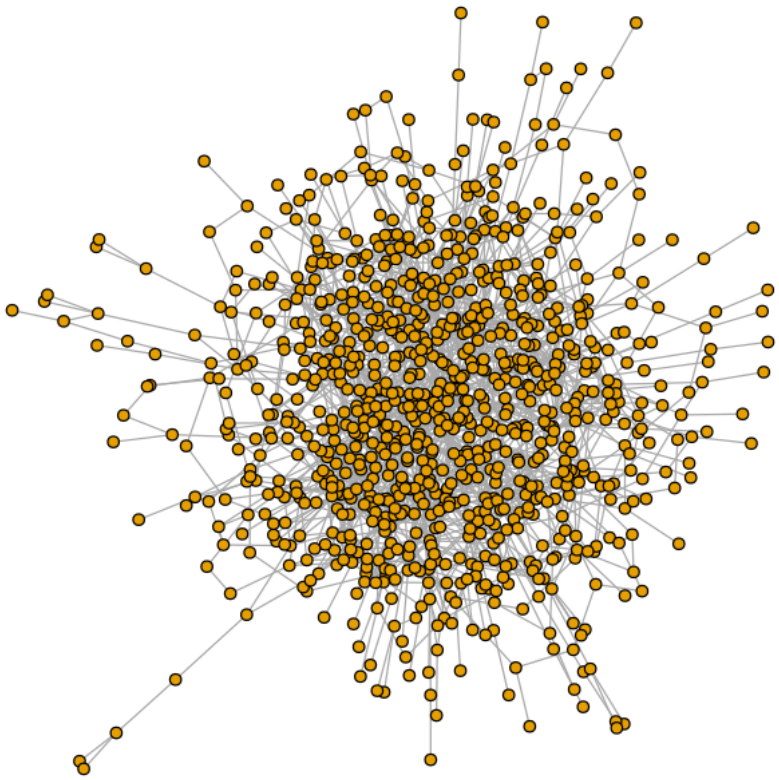
4 1(b)

For each probability, their network was generated 500 times.

[ ]:
```
counter = 0
for (iteration in 1:500){
g1 <- sample_gnp(1000, 0.003, directed = FALSE, loops = FALSE)
```

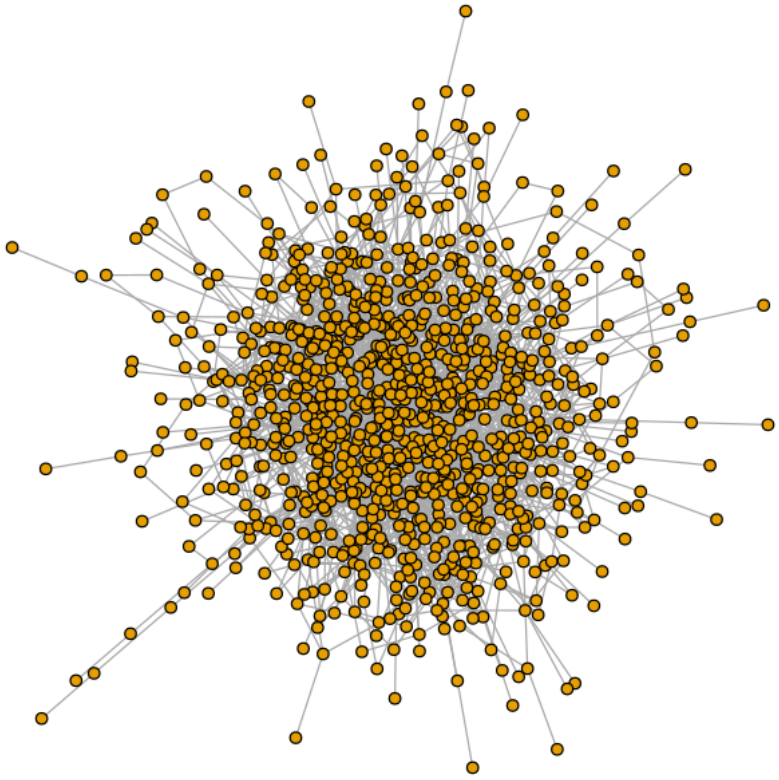
```r
if (is.connected(g1)){
  counter = counter + 1
  }
}
print(sprintf("Connected Probability: %s", counter/500))
g1_components <- clusters(g1)
gcc <- induced.subgraph(g1, which(g1_components$membership == which.
 ↪max(g1_components$csize)))
plot(gcc, vertex.size=3, vertex.label=NA)
print(sprintf("GCC Diameter: %f", diameter(gcc)))
```

```
[1] "Connected Probability: 0"
[1] "GCC Diameter: 13.000000"
```



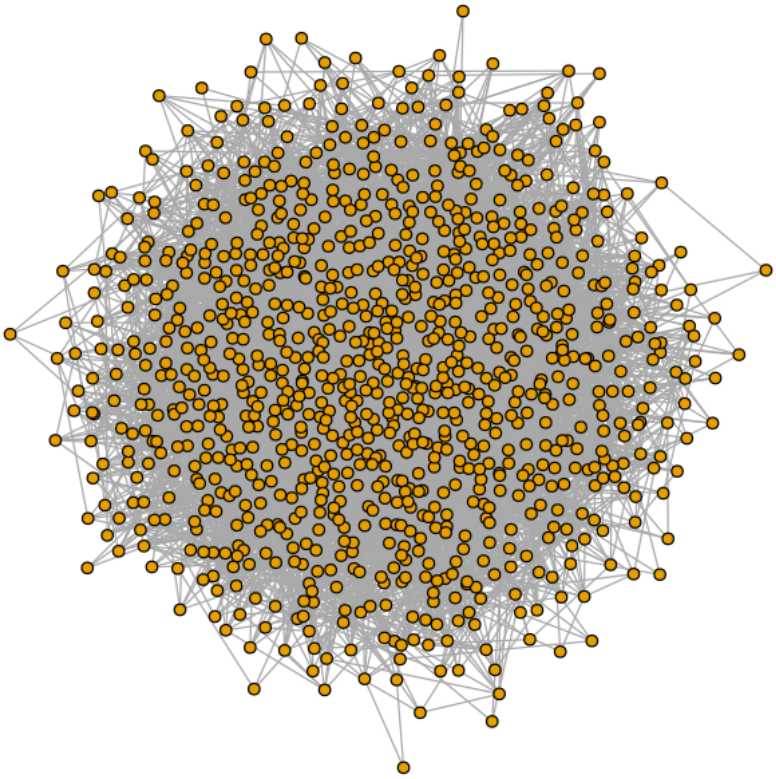
[ ]:
```
counter = 0
for (iteration in 1:500){
g2 <- sample_gnp(1000, 0.004, directed = FALSE, loops = FALSE)
if (is.connected(g2)){
  counter = counter + 1
  }
}
print(sprintf("Connected Probability: %s", counter/500))
g2_components <- clusters(g2)
gcc <- induced.subgraph(g2, which(g2_components$membership == which.
 ↪max(g2_components$csize)))
plot(gcc, vertex.size=3, vertex.label=NA)
print(sprintf("GCC Diameter: %f", diameter(gcc)))
```

```
[1] "Connected Probability: 0"
[1] "GCC Diameter: 11.000000"
```



[ ]:
```
counter = 0
for (iteration in 1:500){
g3 <- sample_gnp(1000, 0.01, directed = FALSE, loops = FALSE)
if (is.connected(g3)){
  counter = counter + 1
  }
}
print(sprintf("Connected Probability: %s", counter/500))
g3_components <- clusters(g3)
gcc <- induced.subgraph(g3, which(g3_components$membership == which.
 ↪max(g3_components$csize)))
plot(gcc, vertex.size=3, vertex.label=NA)
print(sprintf("GCC Diameter: %f", diameter(gcc)))
```

```
[1] "Connected Probability: 0.948"
[1] "GCC Diameter: 5.000000"
```



[ ]:
```r
counter = 0
for (iteration in 1:500){
g4 <- sample_gnp(1000, 0.05, directed = FALSE, loops = FALSE)
if (is.connected(g4)){
  counter = counter + 1
  }
}
print(sprintf("Connected Probability: %s", counter/500))
```

```
[1] "Connected Probability: 1"
```

[ ]:
```
counter = 0
for (iteration in 1:500){
g5 <- sample_gnp(1000, 0.1, directed = FALSE, loops = FALSE)
if (is.connected(g5)){
  counter = counter + 1
  }
}
print(sprintf("Connected Probability: %s", counter/500))
```

```
[1] "Connected Probability: 1"
```

As observed above, for p=0.003 and 0.004, their generated networks were not connected. However, for p=0.01, their networks were connected with a probability of 0.966 and for p=0.05 and 0.1, their networks were always connected.

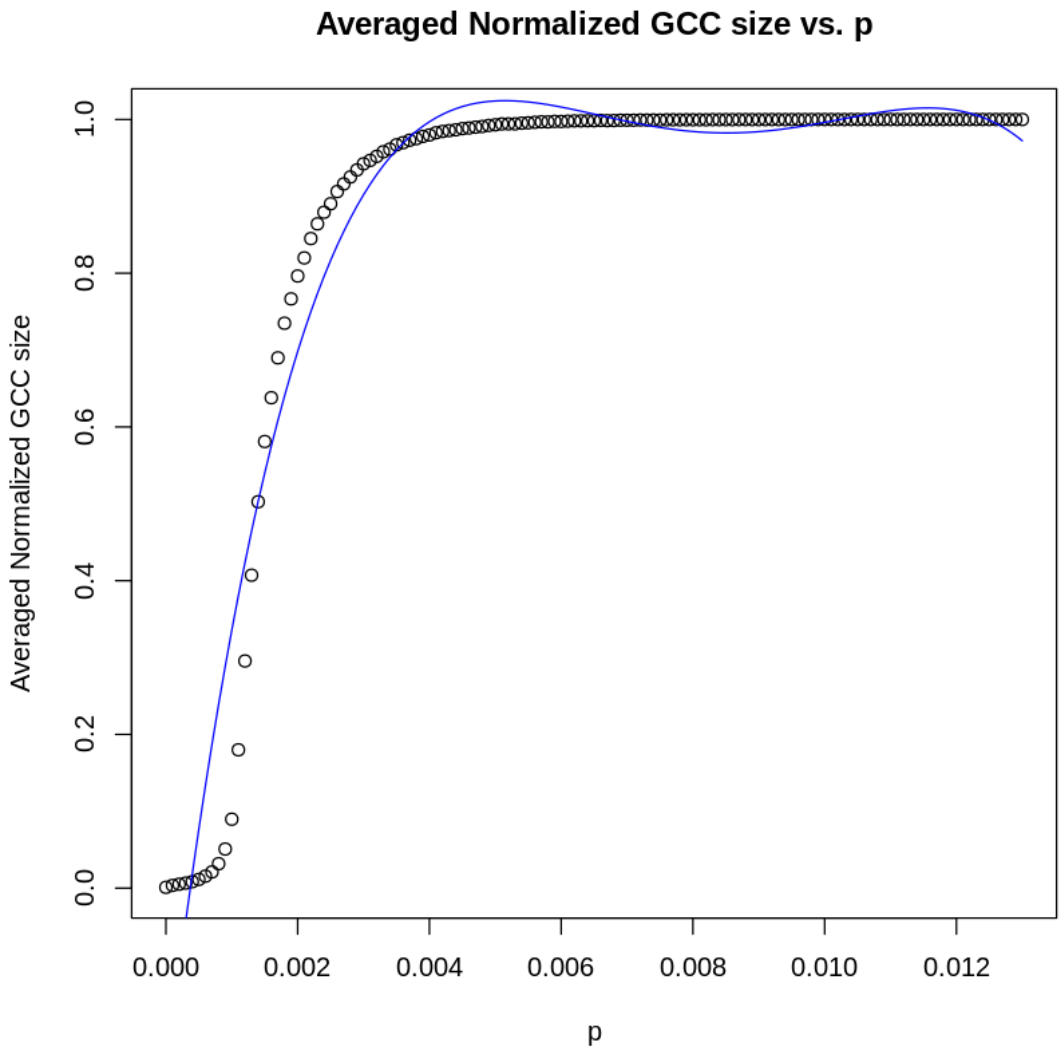
5 1(c)

[ ]:
```
counter = 0
for (iteration in 1:500){
g <- sample_gnp(1000, 0.013, directed = FALSE, loops = FALSE)
if (is.connected(g)){
  counter = counter + 1
  }
}
print(sprintf("Connected Probability: %s", counter/500))
```

```
[1] "Connected Probability: 1"
```

From above, $p_{max} \approx 0.013$.

[ ]:
```
mean_gcc_p = c()
for (p in seq(0, 0.013, by=0.0001)){
  total_gcc_p = c()
  for (iteration in 1:100){
    g <- sample_gnp(1000, p=p, directed = FALSE, loops = FALSE)
    total_gcc_p = c(total_gcc_p, max(components(g)$csize))
    }
  mean_gcc_p = c(mean_gcc_p, mean(total_gcc_p)/1000)
  }
x <- seq(0, 0.013, by=0.0001)
y <- mean_gcc_p
fit <- lm(y ~ poly(x, 4, raw=TRUE))
plot(x, y, main="Averaged Normalized GCC size vs. p", xlab="p", ylab="Averaged␣
 ↪Normalized GCC size")
lines(x, predict(fit, data.frame(x=x)), col='blue')
```



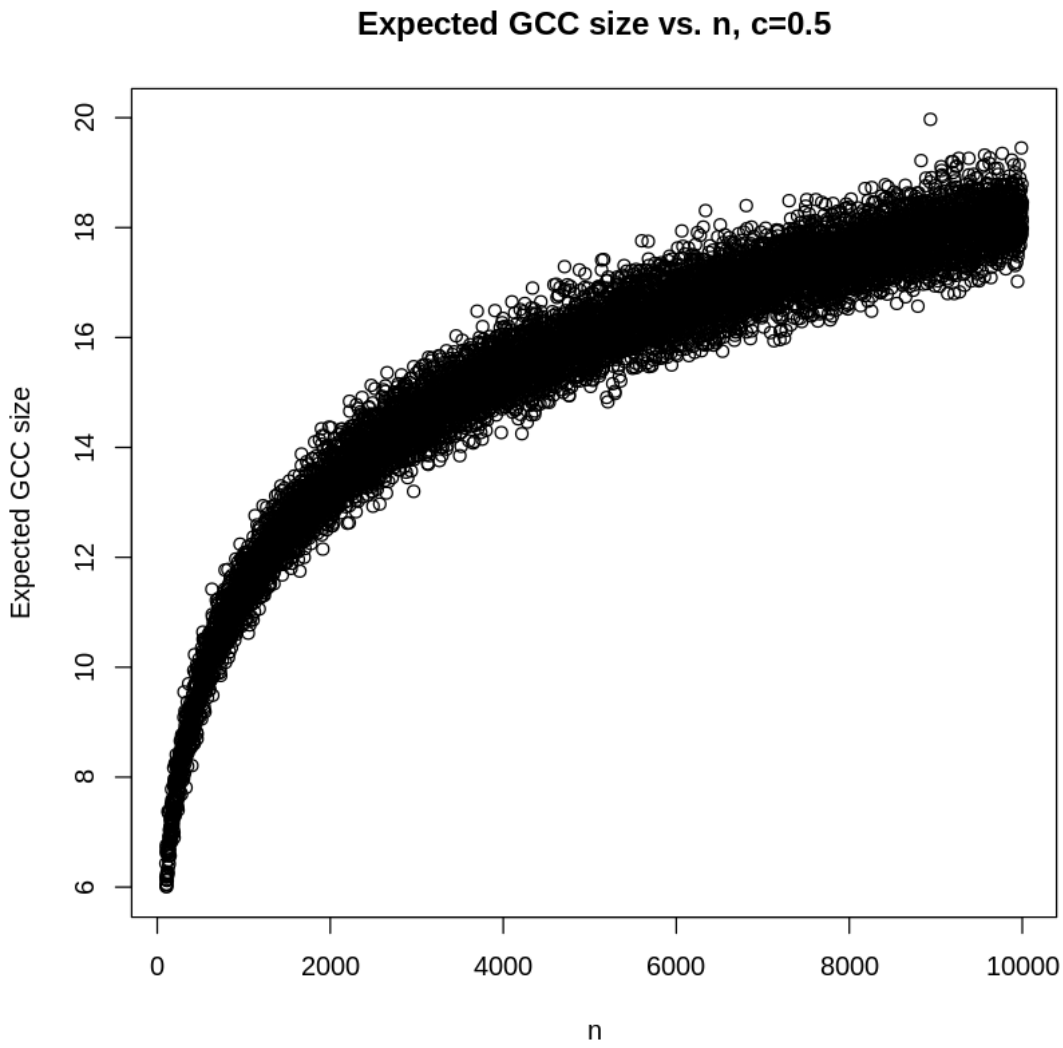
(i, ii) If "emergence" of GCC was defined as the approximate point at which the (slope of normalized GCC size vs. p) rapidly increases from 0, then from above graph, $p \approx 0.001$. It is also observed that the slope tends from a steep slope to 0 when most of the nodes are filled by GCC at $p \approx 0.007$. This conforms with the theoretical values of $p = O(1/n)$ as $1/1000 = 0.001$ and $p = O(ln(n)/n)$ as $ln(1000)/1000 = 0.00691$.

6 1(d)

(i) c=0.5

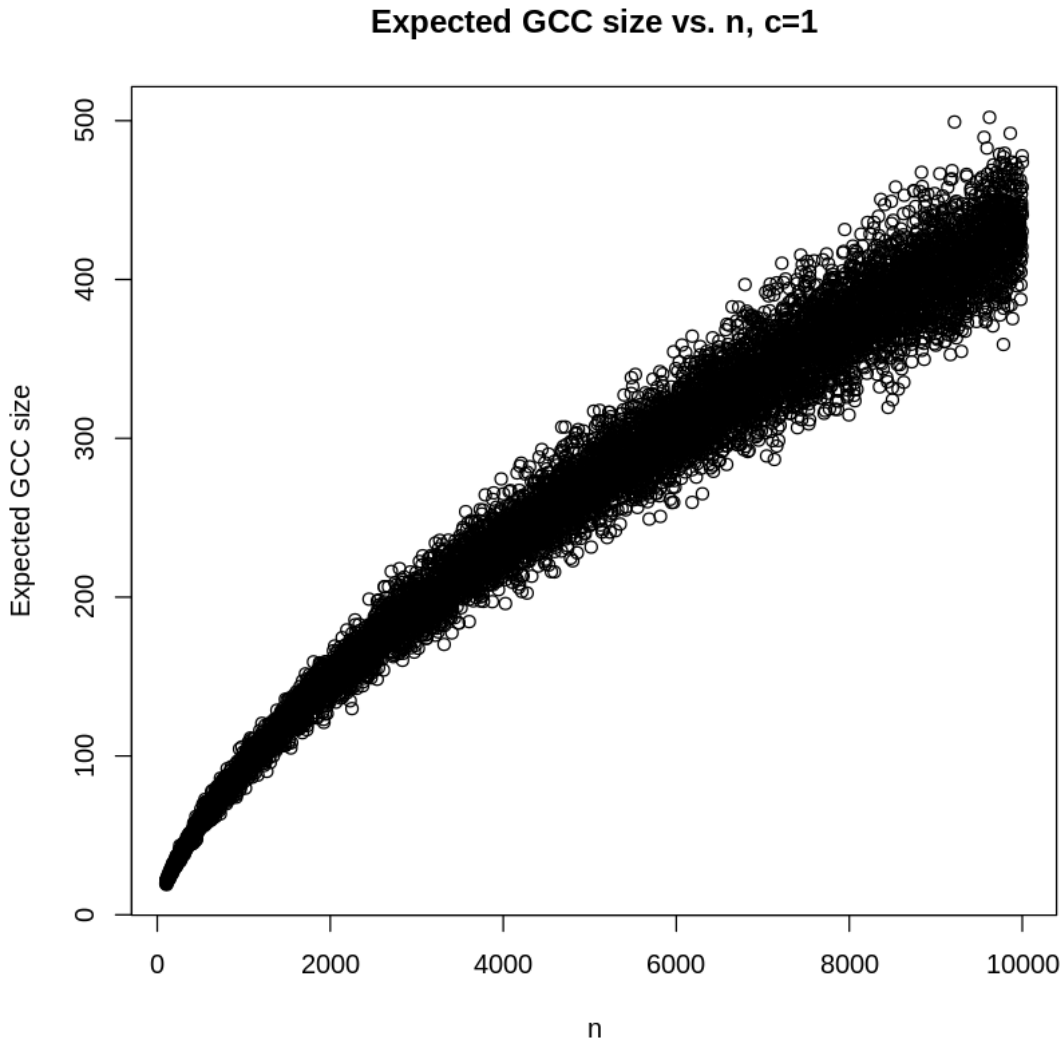
[ ]:
```
mean_gcc_p = c()
for (n in seq(100, 10000, by=1)){
  total_gcc_p = c()
```

```
  for (iteration in 1:100){
    g <- sample_gnp(n=n, p=0.5/n, directed = FALSE, loops = FALSE)
    total_gcc_p = c(total_gcc_p, max(components(g)$csize))
    }
  mean_gcc_p = c(mean_gcc_p, mean(total_gcc_p))
  }
x <- seq(100, 10000, by=1)
y <- mean_gcc_p
plot(x, y, main="Expected GCC size vs. n, c=0.5", xlab="n", ylab="Expected GCC␣
 ↪size")
```



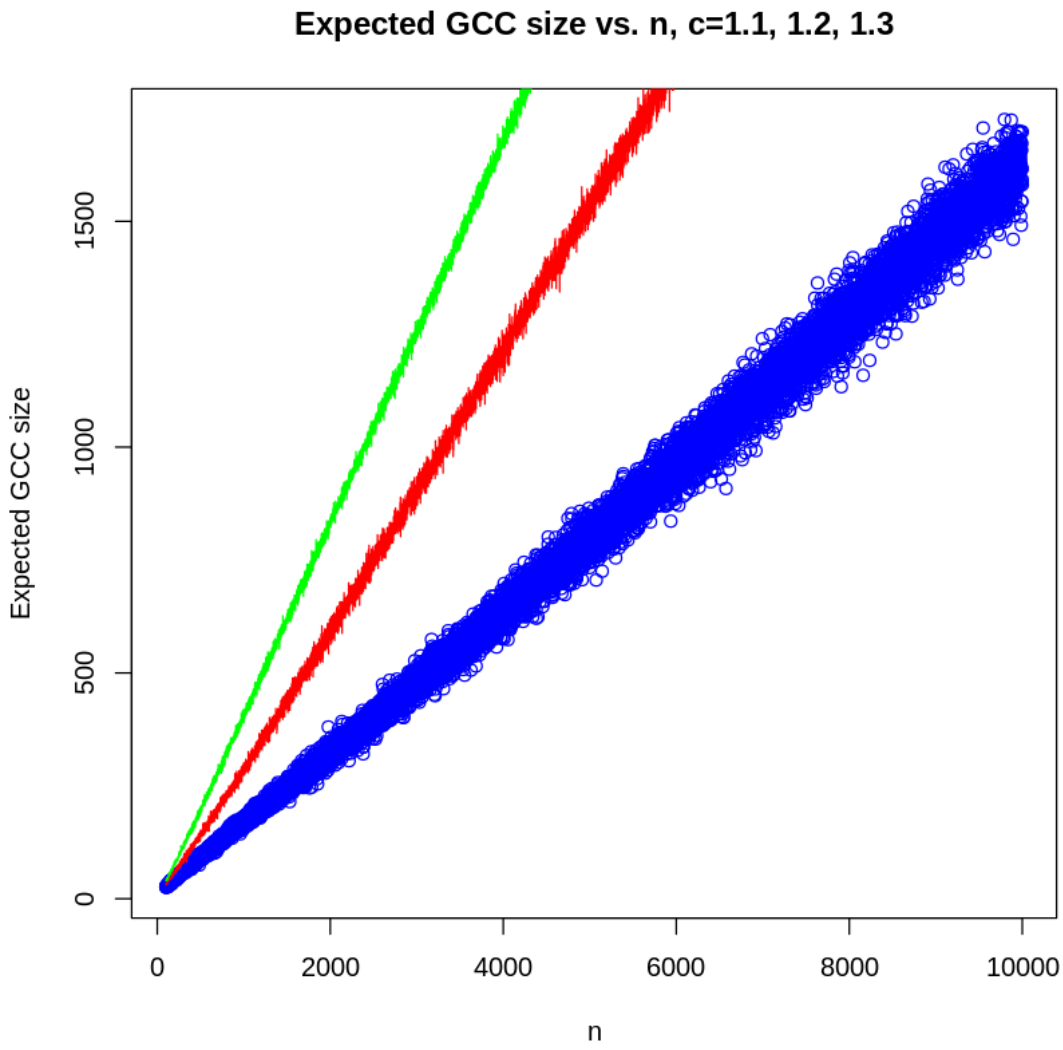
(ii) c=1

[ ]:
```
mean_gcc_p = c()
for (n in seq(100, 10000, by=1)){
  total_gcc_p = c()
  for (iteration in 1:100){
    g <- sample_gnp(n=n, p=1/n, directed = FALSE, loops = FALSE)
    total_gcc_p = c(total_gcc_p, max(components(g)$csize))
    }
  mean_gcc_p = c(mean_gcc_p, mean(total_gcc_p))
  }
x <- seq(100, 10000, by=1)
y <- mean_gcc_p
plot(x, y, main="Expected GCC size vs. n, c=1", xlab="n", ylab="Expected GCC␣
 ↪size")
```



(iii) c=1.1, 1.2, 1.3

[ ]:
```
mean_gcc_p_1 = c()
mean_gcc_p_2 = c()
mean_gcc_p_3 = c()
for (n in seq(100, 10000, by=1)){
  total_gcc_p_1 = c()
  total_gcc_p_2 = c()
  total_gcc_p_3 = c()
  for (iteration in 1:100){
    g1 <- sample_gnp(n=n, p=1.1/n, directed = FALSE, loops = FALSE)
    g2 <- sample_gnp(n=n, p=1.2/n, directed = FALSE, loops = FALSE)
    g3 <- sample_gnp(n=n, p=1.3/n, directed = FALSE, loops = FALSE)
    total_gcc_p_1 = c(total_gcc_p_1, max(components(g1)$csize))
    total_gcc_p_2 = c(total_gcc_p_2, max(components(g2)$csize))
    total_gcc_p_3 = c(total_gcc_p_3, max(components(g3)$csize))
    }
  mean_gcc_p_1 = c(mean_gcc_p_1, mean(total_gcc_p_1))
  mean_gcc_p_2 = c(mean_gcc_p_2, mean(total_gcc_p_2))
  mean_gcc_p_3 = c(mean_gcc_p_3, mean(total_gcc_p_3))
  }


x <- seq(100, 10000, by=1)
y1 <- mean_gcc_p_1
y2 <- mean_gcc_p_2
y3 <- mean_gcc_p_3
plot(x, y1, col="blue", main="Expected GCC size vs. n, c=1.1, 1.2, 1.3",␣
 ↪xlab="n", ylab="Expected GCC size")
lines(x, y2, col="red")
lines(x, y3, col="green")
```



where blue represents c=1.1, red represents c=1.2, and green represents c=1.3.

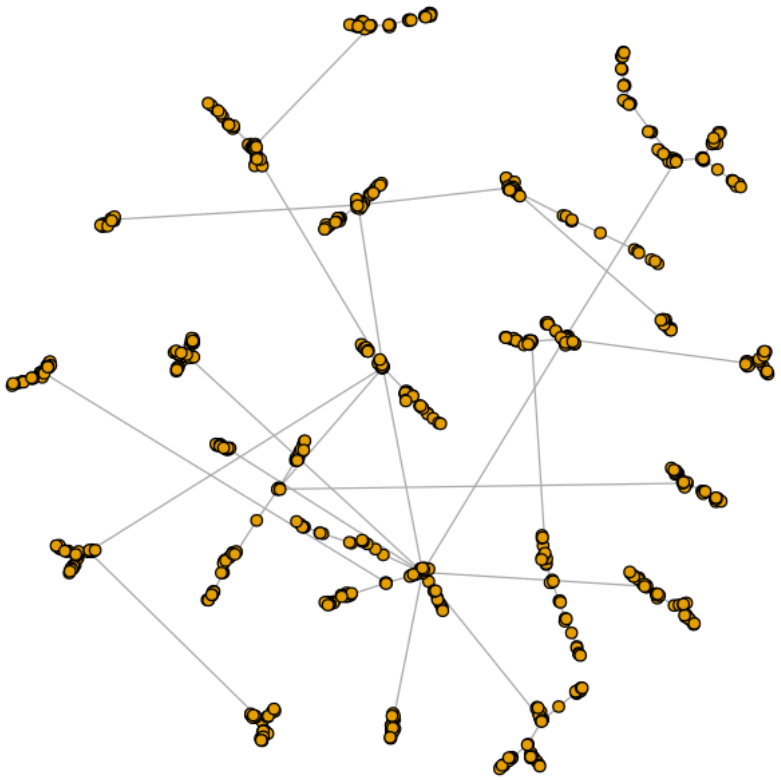
(iv) While GCC size displayed a logarithmic behavior for p=0.5, as the value of p increases, the GCC size curve tends towards a linear correlation.

7 2(a)

[ ]:
```
counter = 0
for (iteration in 1:500){
g1 <- sample_pa(n=1000, m=1, directed = FALSE)
if (is.connected(g1)){
  counter = counter + 1
  }
```

```
}
print(sprintf("Connected Probability: %s", counter/500))
plot(g1, vertex.size=3, vertex.label=NA)
```

```
[1] "Connected Probability: 1"
```

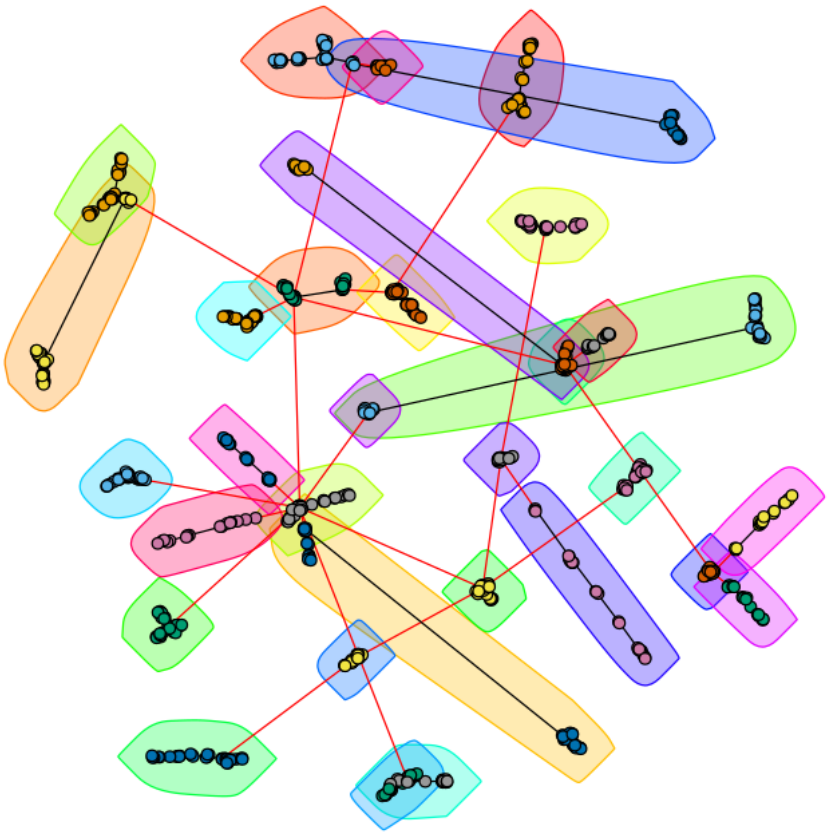


Since this network was tested with 500 iterations and didn't show a single disconnected case, one may infer that the network is always connected.

8 2(b)

[ ]:
```
community_1 <- cluster_fast_greedy(g1)
comm_modularity_1 <- modularity(g1, membership(community_1))
print(sprintf("Modularity: %s", comm_modularity_1))
plot(community_1, g1, vertex.size=3, vertex.label=NA)
```

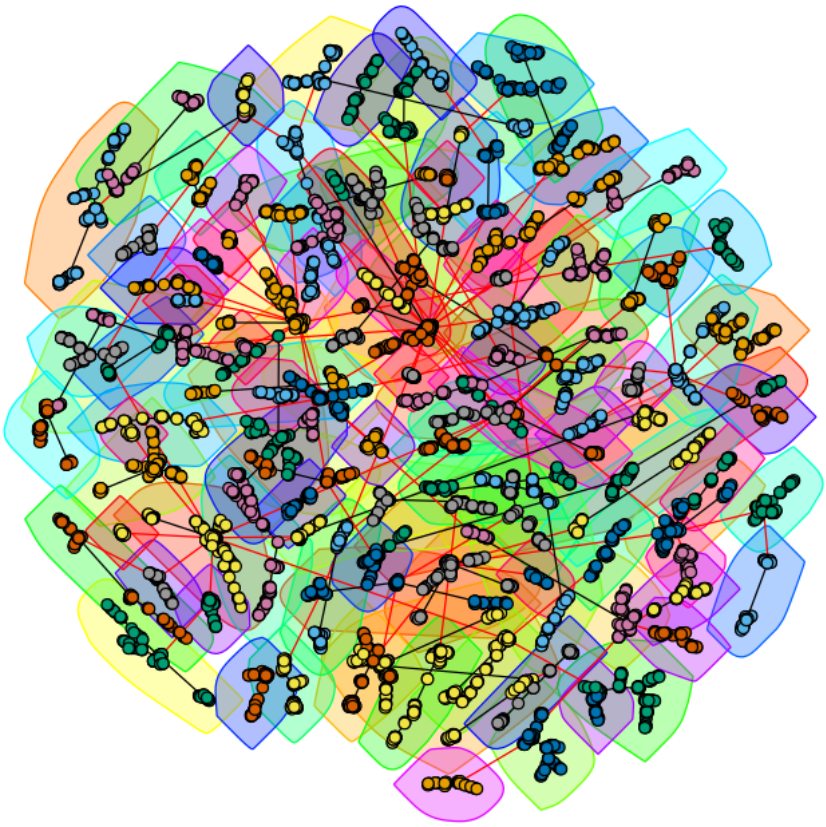
```
[1] "Modularity: 0.934640346051758"
```



9 2(c)

[ ]:
```
counter = 0
for (iteration in 1:500){
g2 <- sample_pa(n=10000, m=1, directed = FALSE)
if (is.connected(g2)){
  counter = counter + 1
  }
}
community <- cluster_fast_greedy(g2)
comm_modularity <- modularity(g2, membership(community))
print(sprintf("Modularity: %s", comm_modularity))
plot(community, g2, vertex.size=3, vertex.label=NA)
```

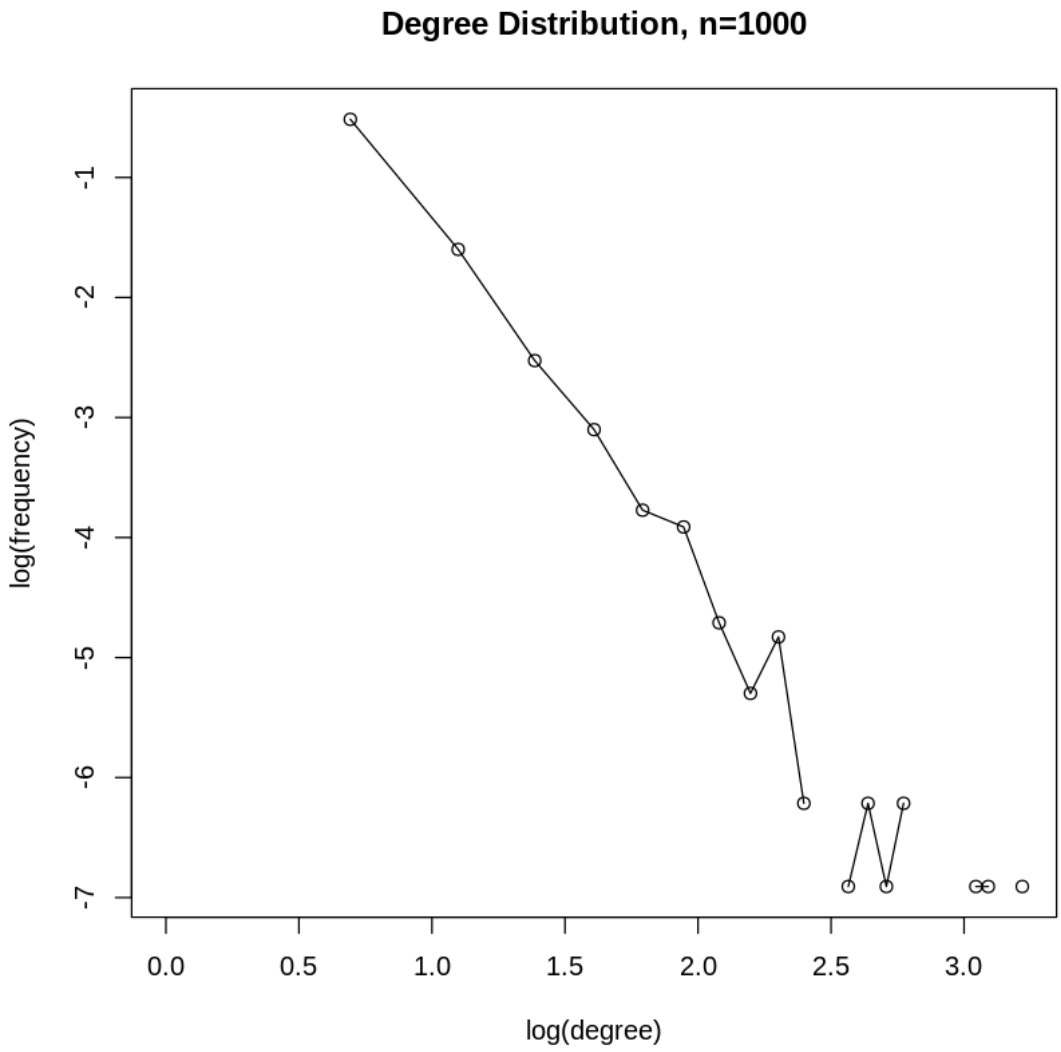
```
[1] "Modularity: 0.978362387693915"
```



With 10,000 nodes, the modularity has increased.

10 2(d)

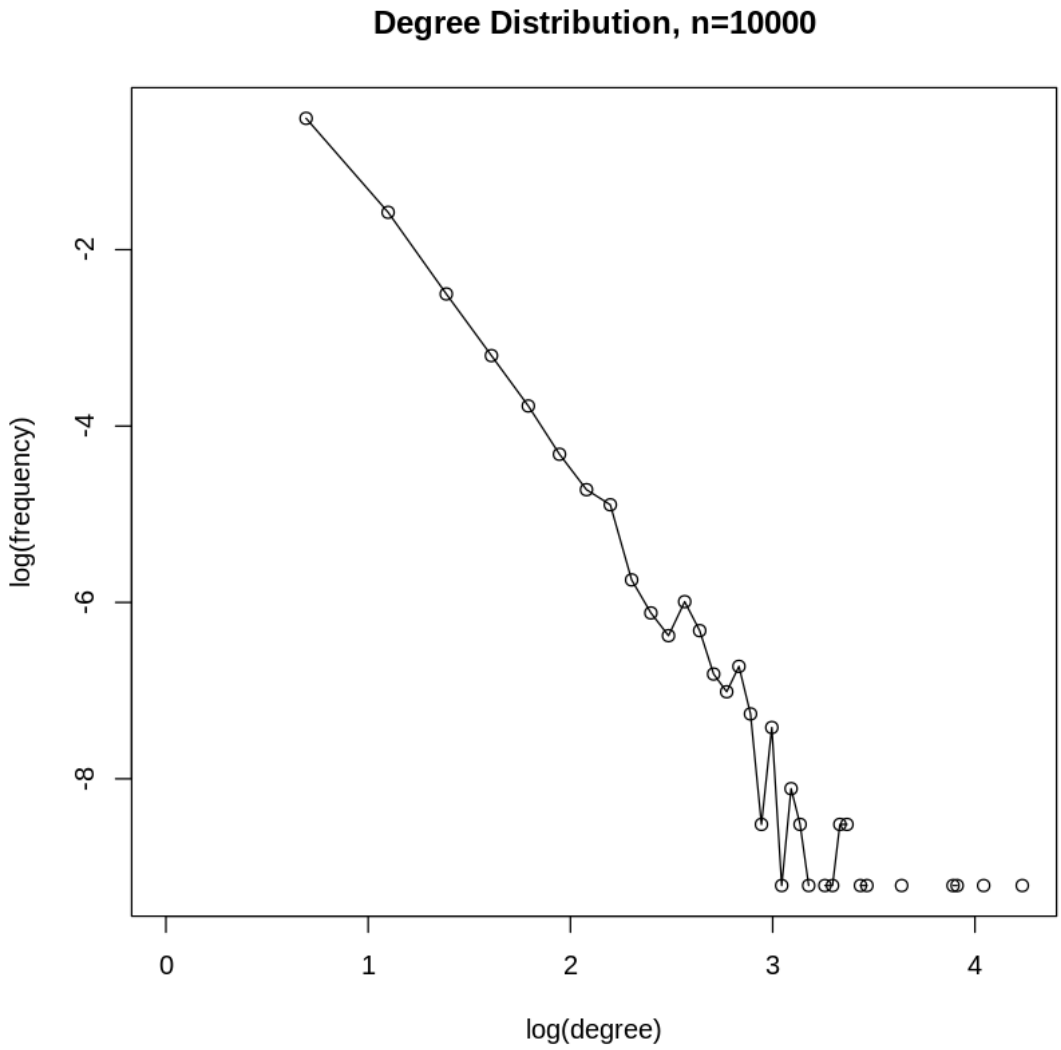
[ ]:
```
#n=1000
degree1 <- log(c(1:length(degree.distribution(g1))))
dist1 <- log(degree.distribution(g1))
df1 <- data.frame(x=degree1, y=dist1)
plot(df1$x, df1$y, main='Degree Distribution, n=1000', xlab = "log(degree)",␣
 ↪ylab = "log(frequency)", type='o')
```



[ ]:
```
df1_new <- df1
df1_new[is.na(df1_new) | df1_new == "Inf"] <- NA
df1_new[is.na(df1_new) | df1_new == "-Inf"] <- NA
lm1 = lm(y~x, data = df1_new)
coef(lm1)[2]
```

x: -2.7690210303281

[ ]:
```
#n=10,000
degree2 <- log(c(1:length(degree.distribution(g2))))
dist2 <- log(degree.distribution(g2))
df2 <- data.frame(x=degree2, y=dist2)
plot(df2$x, df2$y, main='Degree Distribution, n=10000', xlab = "log(degree)",␣
↪ylab = "log(frequency)", type='o')
```



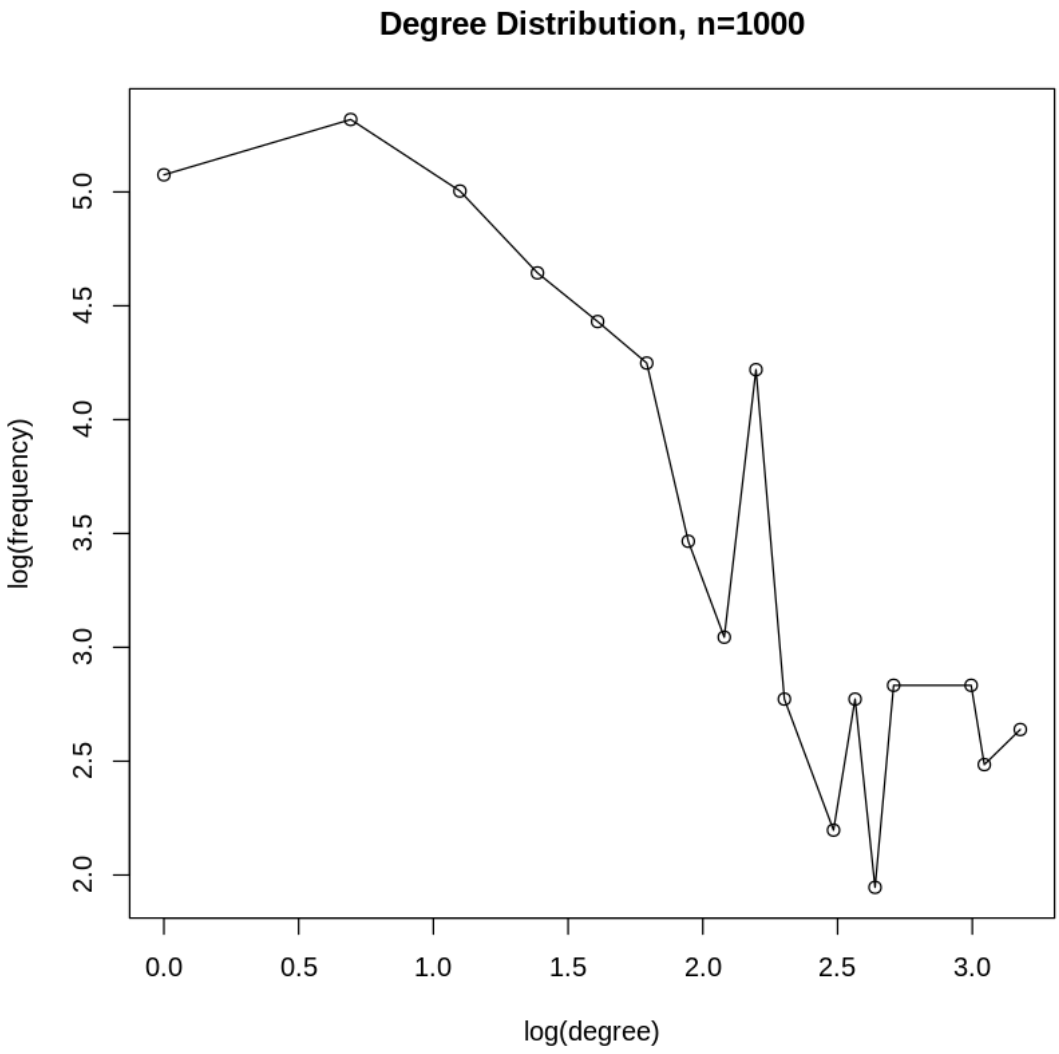
[ ]:
```
df2_new <- df2
df2_new[is.na(df2_new) | df2_new == "Inf"] <- NA
df2_new[is.na(df2_new) | df2_new == "-Inf"] <- NA
lm2 = lm(y~x, data = df2_new)
coef(lm2)[2]
```

x: -2.84016452055787

Slope of plot is approximately -2.75 for n=1000 and -2.88 for n=10000, indicating a slightly more negative value with a larger number of nodes.

11 2(e)

[ ]:
```
#n=1000
node_neighbor_degree = c()
for (i in 1:vcount(g1)){
  node = sample(vcount(g1), 1)
  node_neighbors = neighbors(g1, node)
  if (length(node_neighbors) != 0){
    node_neighbor = sample(length(node_neighbors), 1)
    node_neighbor_degree = c(node_neighbor_degree, degree(g1,␣
 ↪node_neighbors[node_neighbor]))
    }
}
df3 <- data.frame(table(node_neighbor_degree))
df3[,1] <- log(as.numeric(as.character( df3[, 1] )))
df3[,2] <- log(as.numeric(as.character( df3[, 2] )))
plot(df3$node_neighbor_degree, df3$Freq, main='Degree Distribution, n=1000',␣
 ↪xlab = "log(degree)", ylab = "log(frequency)", type='o')
```

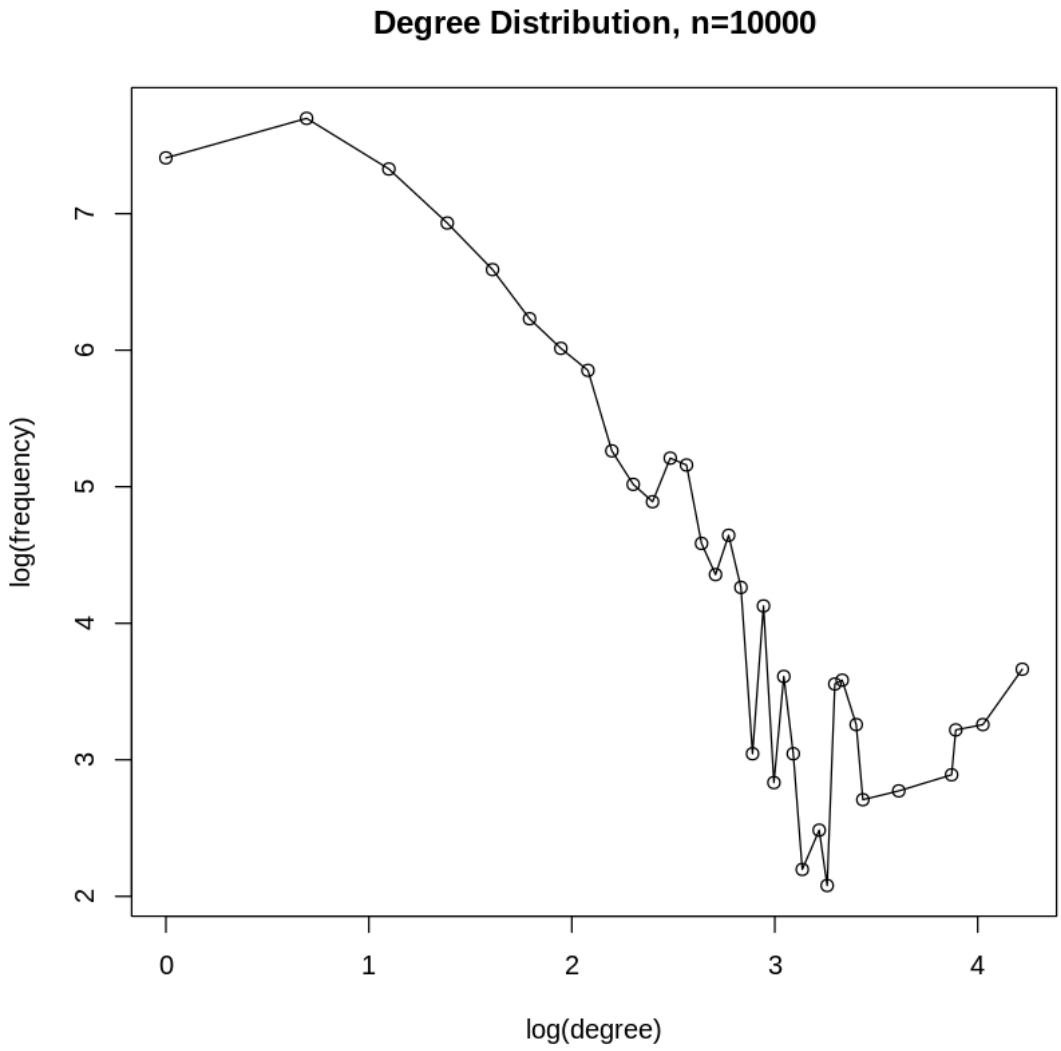


[ ]:
```
df3_new <- df3
df3_new[is.na(df3_new) | df3_new == "Inf"] <- NA
df3_new[is.na(df3_new) | df3_new == "-Inf"] <- NA
lm3 = lm(Freq~node_neighbor_degree, data = df3_new)
coef(lm3)[2]
```

node_neighbor_degree: -1.15297916236204

[ ]:
```
#n=10000
node_neighbor_degree = c()
for (i in 1:vcount(g2)){
  node = sample(vcount(g2), 1)
  node_neighbors = neighbors(g2, node)
```

```
  if (length(node_neighbors) != 0){
    node_neighbor = sample(length(node_neighbors), 1)
    node_neighbor_degree = c(node_neighbor_degree, degree(g2,␣
 ↪node_neighbors[node_neighbor]))
    }
}
df4 <- data.frame(table(node_neighbor_degree))
df4[,1] <- log(as.numeric(as.character( df4[, 1] )))
df4[,2] <- log(as.numeric(as.character( df4[, 2] )))
plot(df4$node_neighbor_degree, df4$Freq, main='Degree Distribution, n=10000',␣
 ↪xlab = "log(degree)", ylab = "log(frequency)", type='o')
```



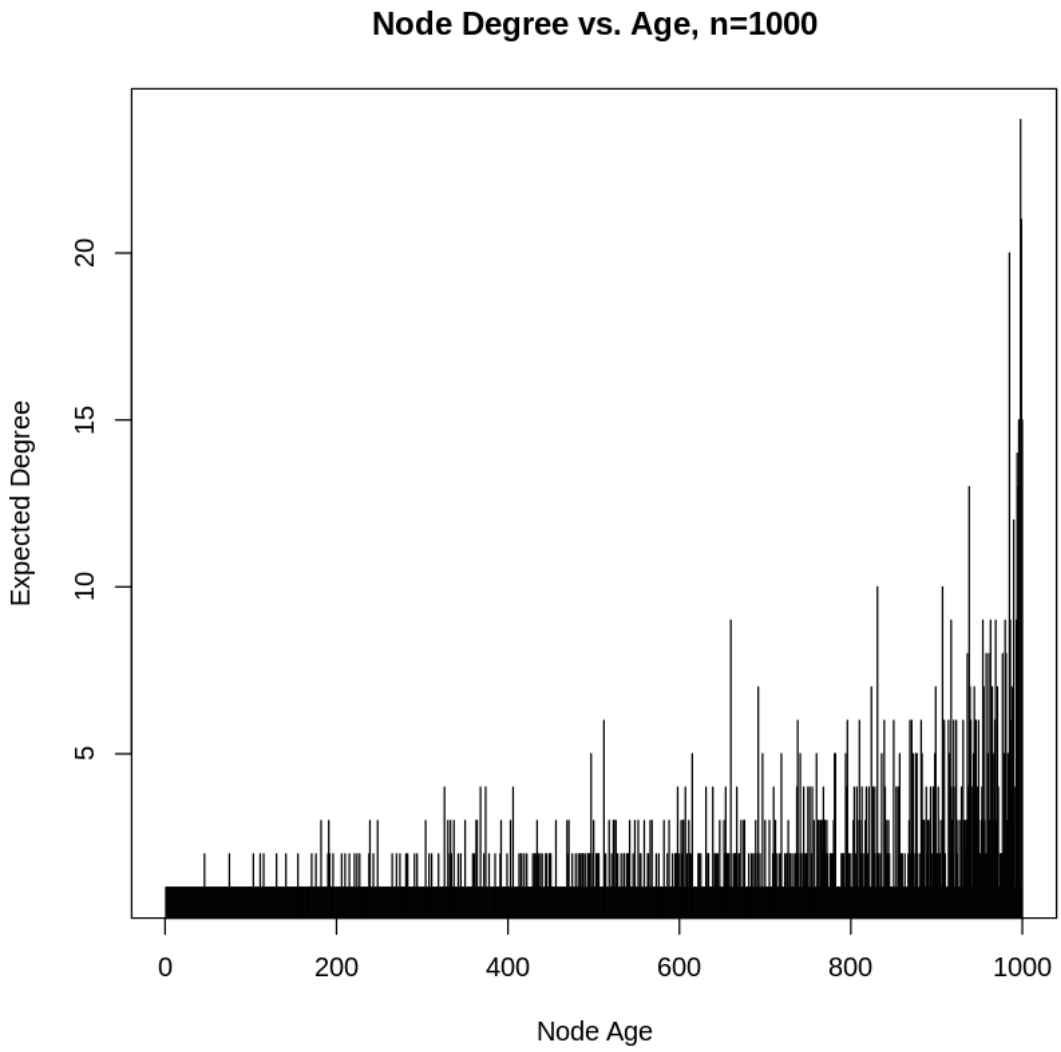
[ ]:
```
df4_new <- df4
df4_new[is.na(df4_new) | df4_new == "Inf"] <- NA
df4_new[is.na(df4_new) | df4_new == "-Inf"] <- NA
lm4 = lm(Freq~node_neighbor_degree, data = df4_new)
coef(lm4)[2]
```

node_neighbor_degree: -1.5368263617025

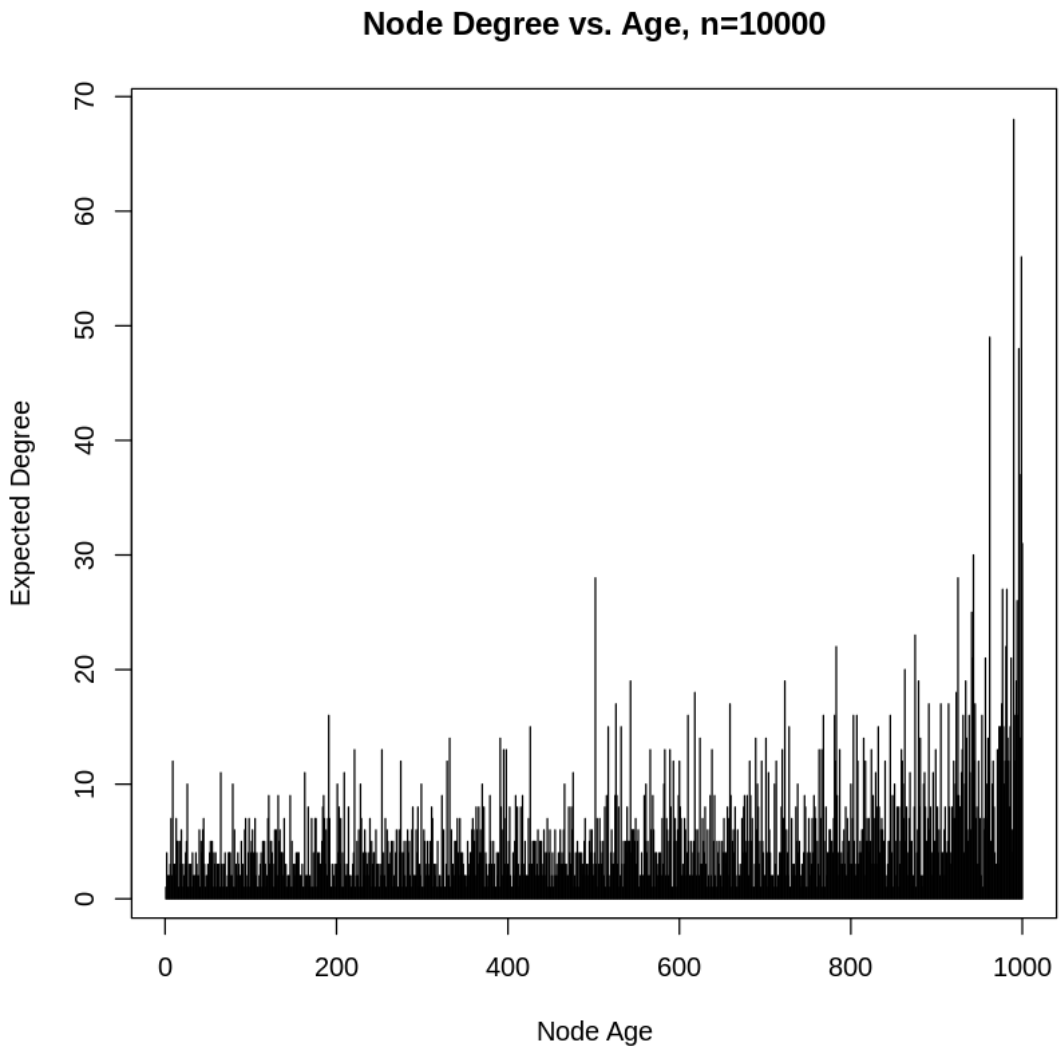
The distribution is indeed linear in the log-log scale but its correlation is more difficult to observe with less number of nodes. Furthermore, the latter degree distributions using neigboring nodes have higher count frequencies because the probability of picking a neighboring node is higher if we start with a random node and then randomly pick their neighbor.

12 2(f)

[ ]:
```
node_degree <- degree(g1)[1:1000]
node_age <- rev(node_degree)
plot(node_age, main="Node Degree vs. Age, n=1000",xlab="Node␣
 ↪Age",ylab="Expected Degree", type='h')
```



[ ]:
```
node_degree <- degree(g2)[1:1000]
node_age <- rev(node_degree)
plot(node_age, main="Node Degree vs. Age, n=10000",xlab="Node␣
  ↪Age",ylab="Expected Degree", type='h')
```



13 2(g)

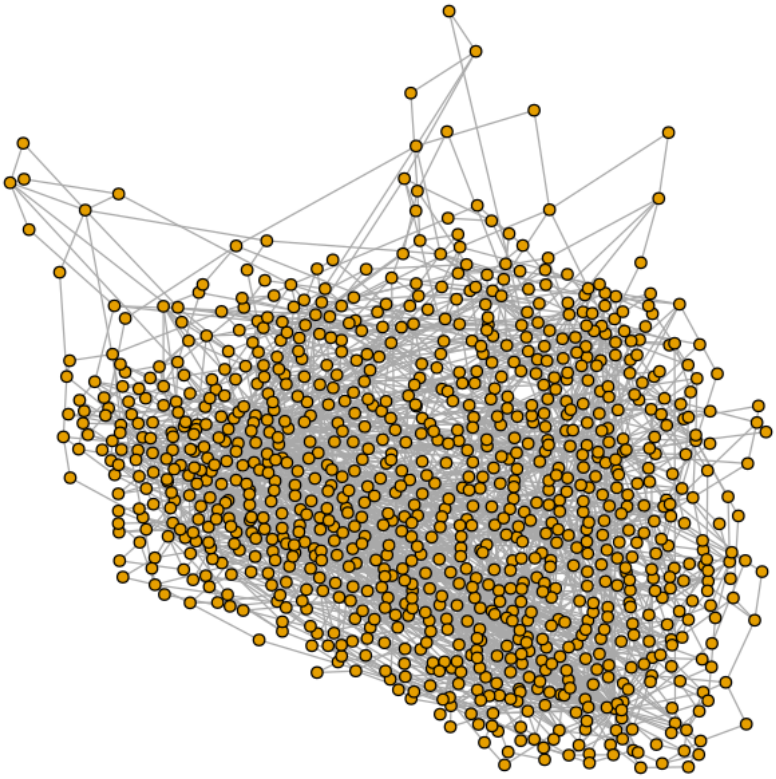
m=2

[ ]:
```
counter = 0
for (iteration in 1:500){
g1 <- sample_pa(n=1000, m=2, directed = FALSE)
if (is.connected(g1)){
  counter = counter + 1
  }
}
print(sprintf("Connected Probability: %s", counter/500))
plot(g1, vertex.size=3, vertex.label=NA)
```

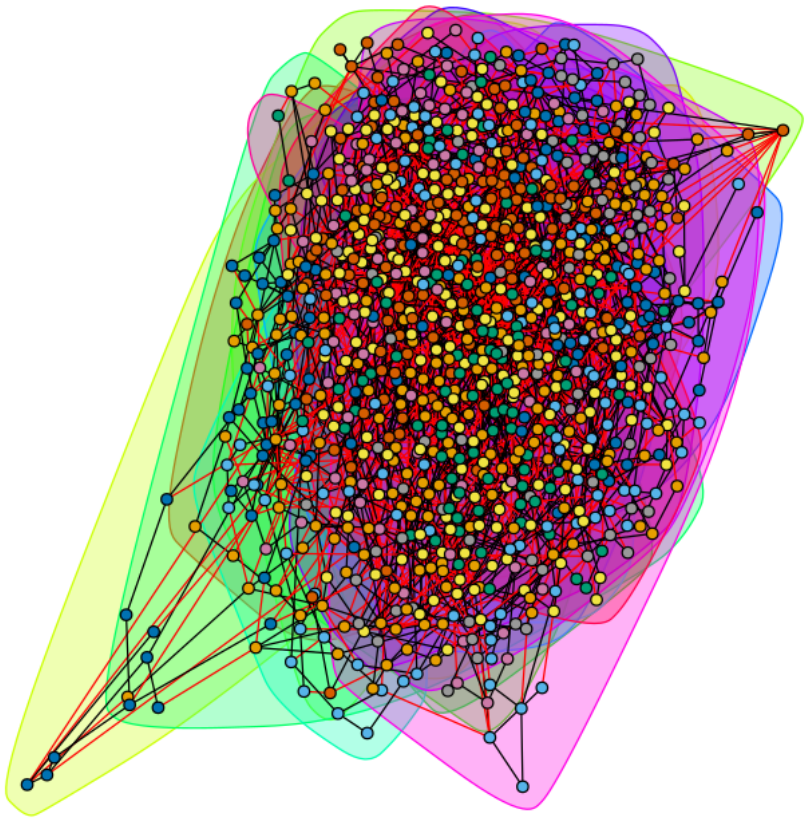
```
community_1 <- cluster_fast_greedy(g1)
comm_modularity_1 <- modularity(g1, membership(community_1))
print(sprintf("Modularity: %s", comm_modularity_1))
plot(community_1, g1, vertex.size=3, vertex.label=NA)
counter = 0
for (iteration in 1:500){
g2 <- sample_pa(n=10000, m=1, directed = FALSE)
if (is.connected(g2)){
  counter = counter + 1
  }
}
community <- cluster_fast_greedy(g2)
comm_modularity <- modularity(g2, membership(community))
print(sprintf("Modularity: %s", comm_modularity))
plot(community, g2, vertex.size=3, vertex.label=NA)

#n=1000
degree1 <- log(c(1:length(degree.distribution(g1))))
dist1 <- log(degree.distribution(g1))
df1 <- data.frame(x=degree1, y=dist1)
plot(df1$x, df1$y, main='Degree Distribution, n=1000', xlab = "log(degree)",␣
 ↪ylab = "log(frequency)", type='o')
df1_new <- df1
df1_new[is.na(df1_new) | df1_new == "Inf"] <- NA
df1_new[is.na(df1_new) | df1_new == "-Inf"] <- NA
lm1 = lm(y~x, data = df1_new)
coef(lm1)[2]

#n=10,000
degree2 <- log(c(1:length(degree.distribution(g2))))
dist2 <- log(degree.distribution(g2))
df2 <- data.frame(x=degree2, y=dist2)
plot(df2$x, df2$y, main='Degree Distribution, n=10000', xlab = "log(degree)",␣
 ↪ylab = "log(frequency)", type='o')
df2_new <- df2
df2_new[is.na(df2_new) | df2_new == "Inf"] <- NA
df2_new[is.na(df2_new) | df2_new == "-Inf"] <- NA
lm2 = lm(y~x, data = df2_new)
coef(lm2)[2]

#n=1000
node_neighbor_degree = c()
for (i in 1:vcount(g1)){
  node = sample(vcount(g1), 1)
  node_neighbors = neighbors(g1, node)
  if (length(node_neighbors) != 0){
    node_neighbor = sample(length(node_neighbors), 1)
```

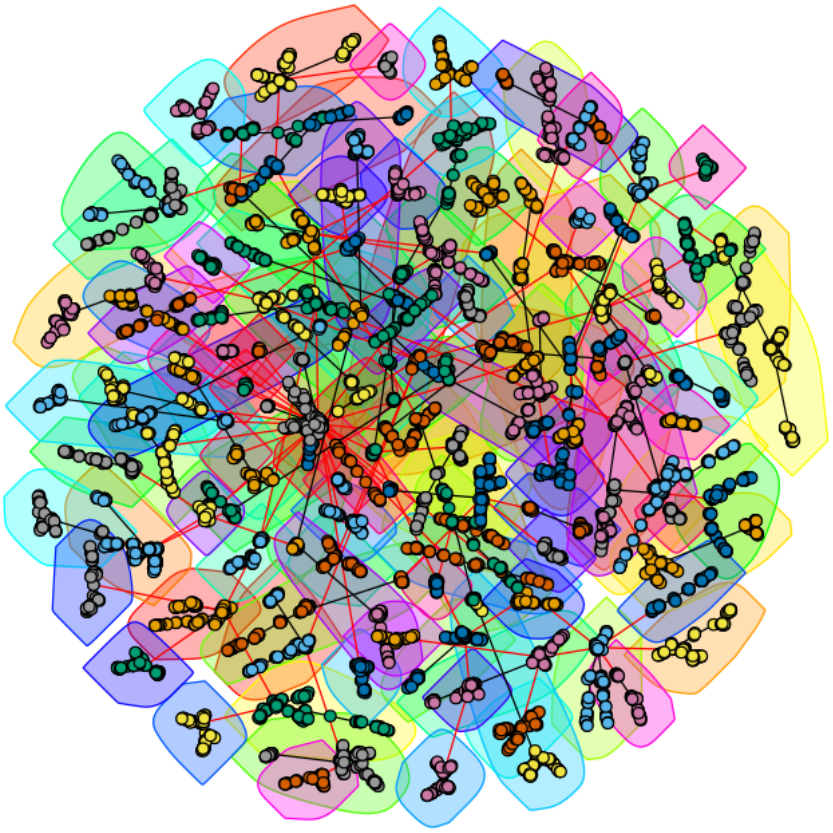
```
    node_neighbor_degree = c(node_neighbor_degree, degree(g1,␣
 ↪node_neighbors[node_neighbor]))
    }
}
df3 <- data.frame(table(node_neighbor_degree))
df3[,1] <- log(as.numeric(as.character( df3[, 1] )))
df3[,2] <- log(as.numeric(as.character( df3[, 2] )))
plot(df3$node_neighbor_degree, df3$Freq, main='Degree Distribution, n=1000',␣
 ↪xlab = "log(degree)", ylab = "log(frequency)", type='o')
df3_new <- df3
df3_new[is.na(df3_new) | df3_new == "Inf"] <- NA
df3_new[is.na(df3_new) | df3_new == "-Inf"] <- NA
lm3 = lm(Freq~node_neighbor_degree, data = df3_new)
coef(lm3)[2]

#n=10000
node_neighbor_degree = c()
for (i in 1:vcount(g2)){
  node = sample(vcount(g2), 1)
  node_neighbors = neighbors(g2, node)
  if (length(node_neighbors) != 0){
    node_neighbor = sample(length(node_neighbors), 1)
    node_neighbor_degree = c(node_neighbor_degree, degree(g2,␣
 ↪node_neighbors[node_neighbor]))
    }
}
df4 <- data.frame(table(node_neighbor_degree))
df4[,1] <- log(as.numeric(as.character( df4[, 1] )))
df4[,2] <- log(as.numeric(as.character( df4[, 2] )))
plot(df4$node_neighbor_degree, df4$Freq, main='Degree Distribution, n=10000',␣
 ↪xlab = "log(degree)", ylab = "log(frequency)", type='o')
df4_new <- df4
df4_new[is.na(df4_new) | df4_new == "Inf"] <- NA
df4_new[is.na(df4_new) | df4_new == "-Inf"] <- NA
lm4 = lm(Freq~node_neighbor_degree, data = df4_new)
coef(lm4)[2]

node_degree <- degree(g1)[1:1000]
node_age <- rev(node_degree)
plot(node_age, main="Node Degree vs. Age, n=1000",xlab="Node␣
 ↪Age",ylab="Expected Degree", type='h')
node_degree <- degree(g2)[1:1000]
node_age <- rev(node_degree)
plot(node_age, main="Node Degree vs. Age, n=10000",xlab="Node␣
 ↪Age",ylab="Expected Degree", type='h')
```

```
[1] "Connected Probability: 1"
[1] "Modularity: 0.524352126587477"
```

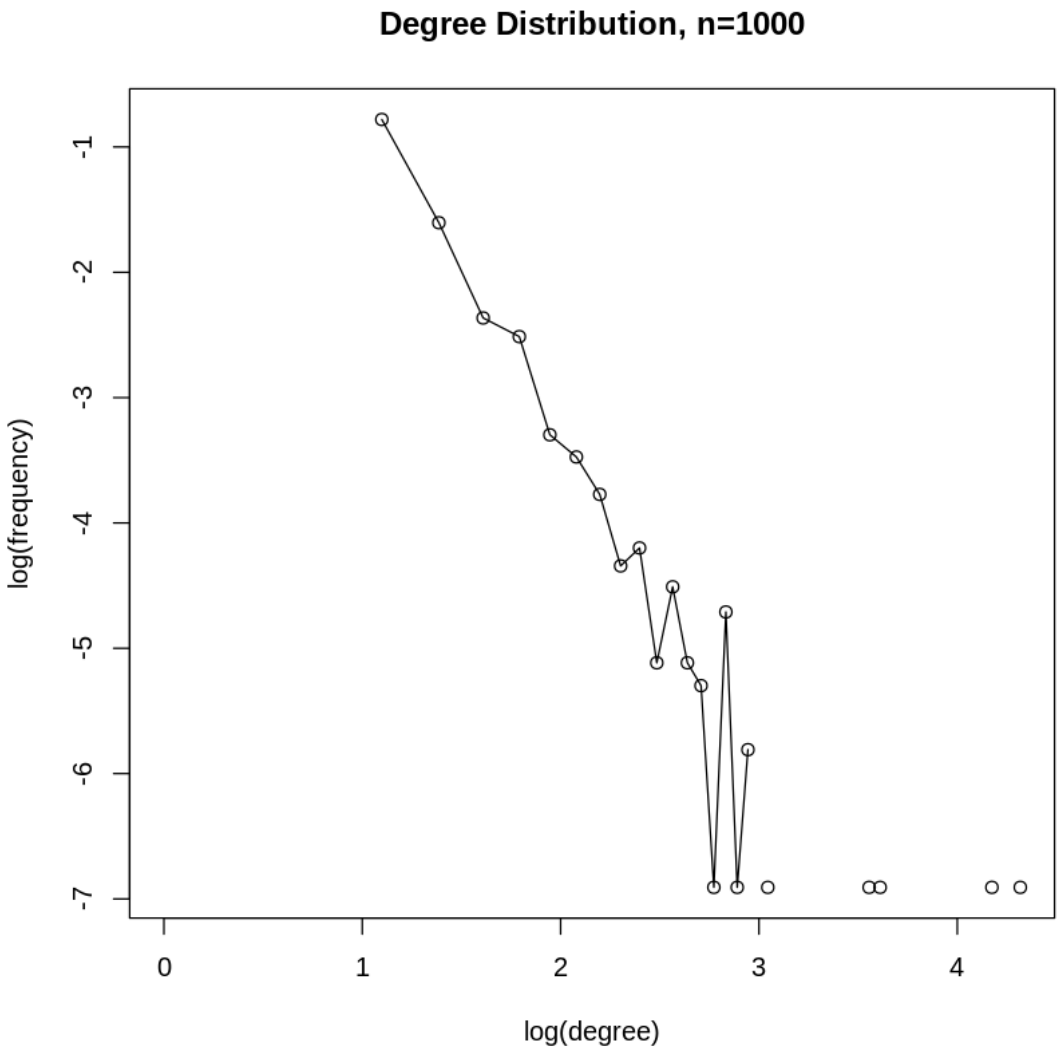


```
[1] "Modularity: 0.978083366892545"
```

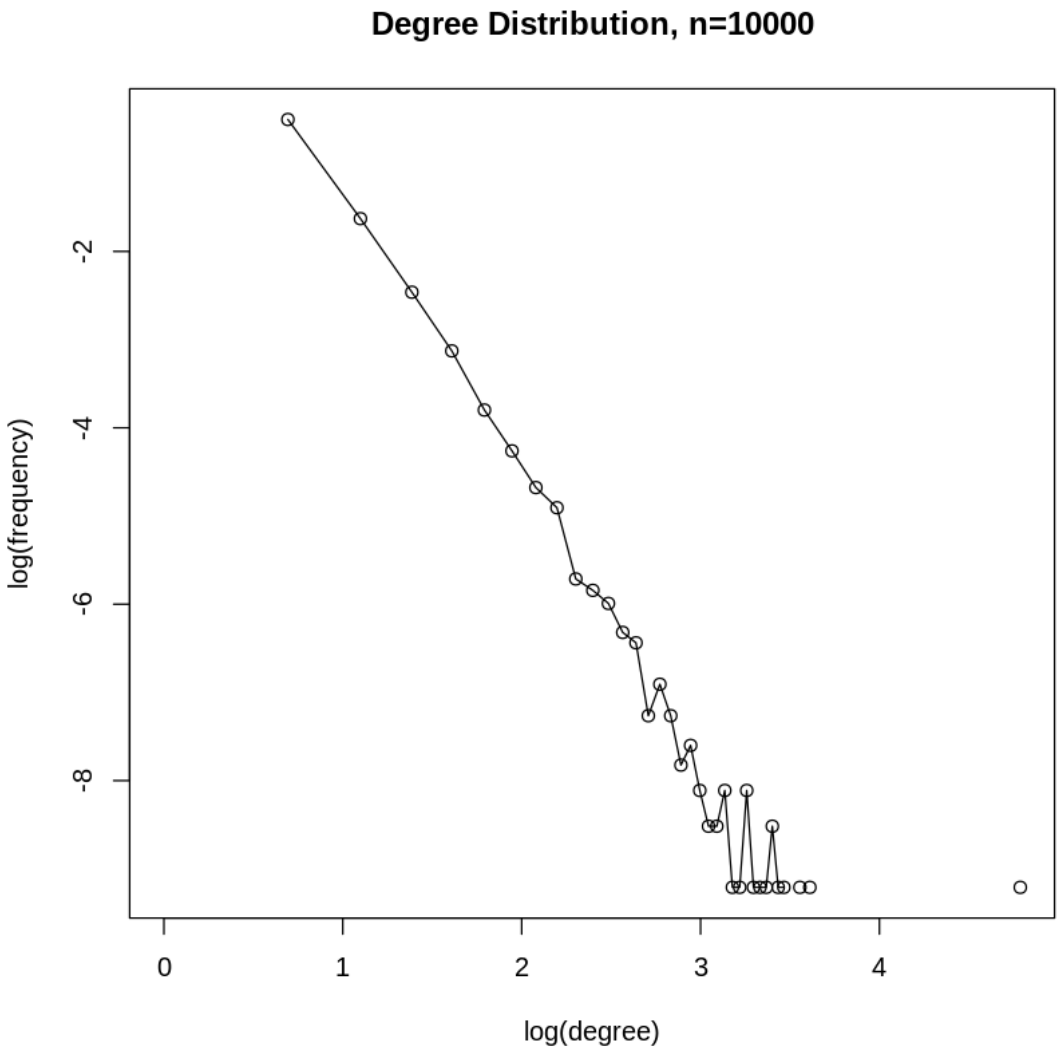




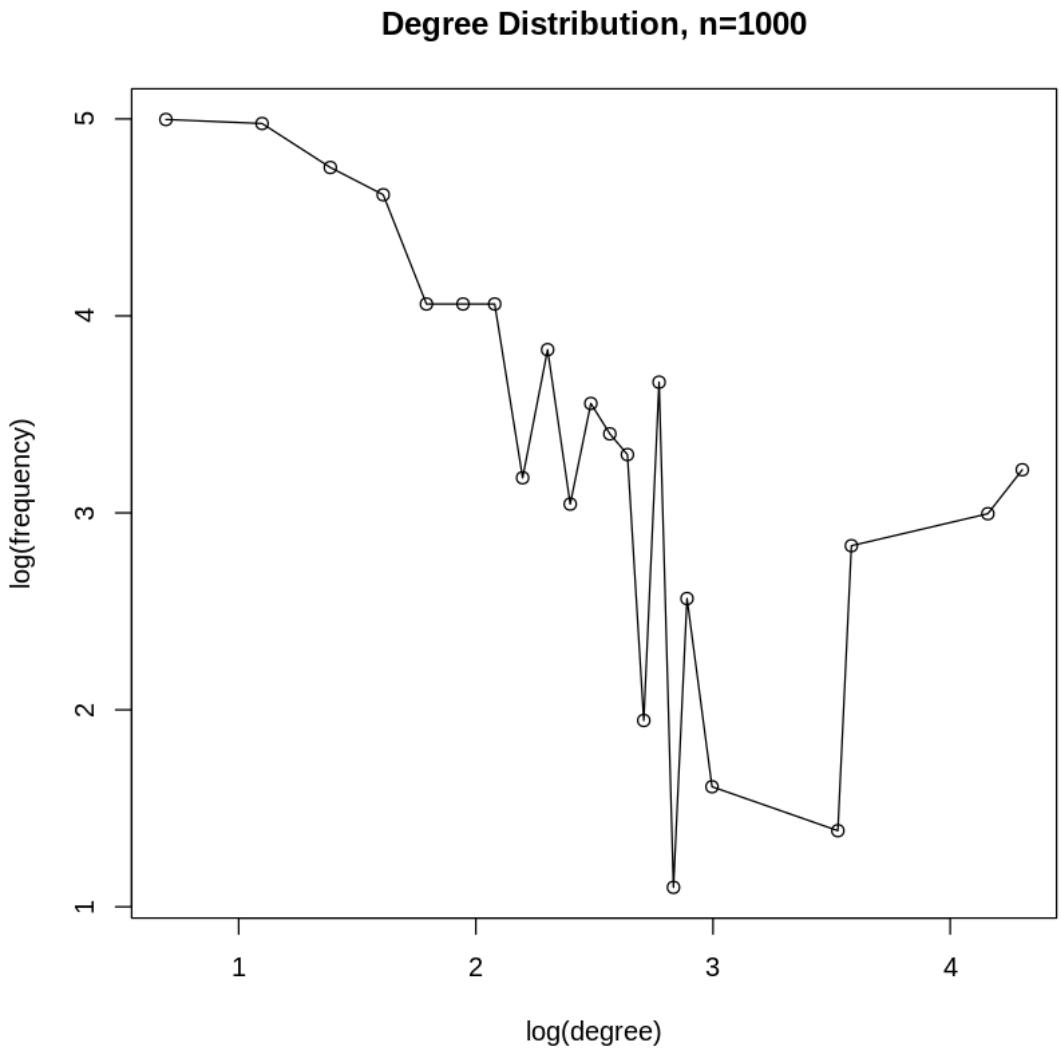
x: -2.10050283706177



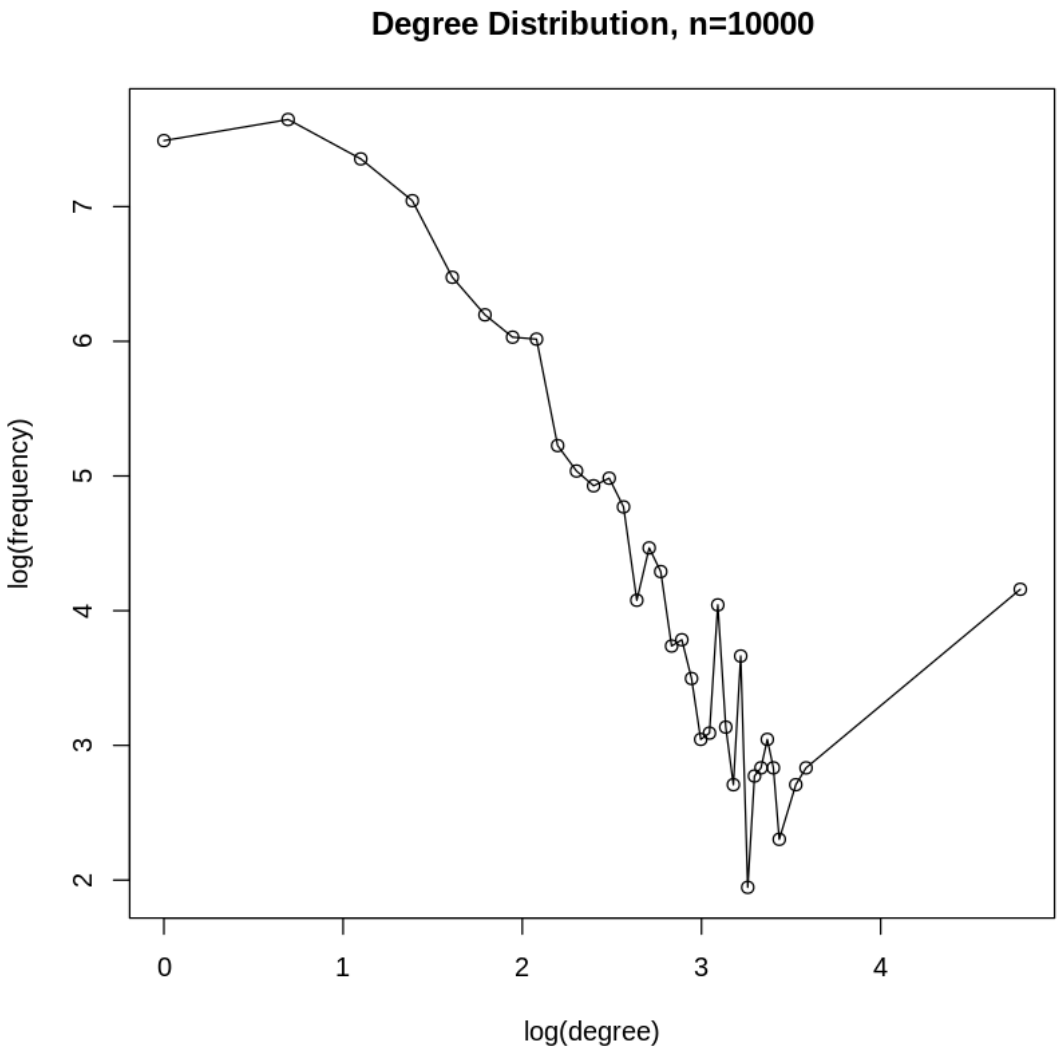
x: -2.85856206000213

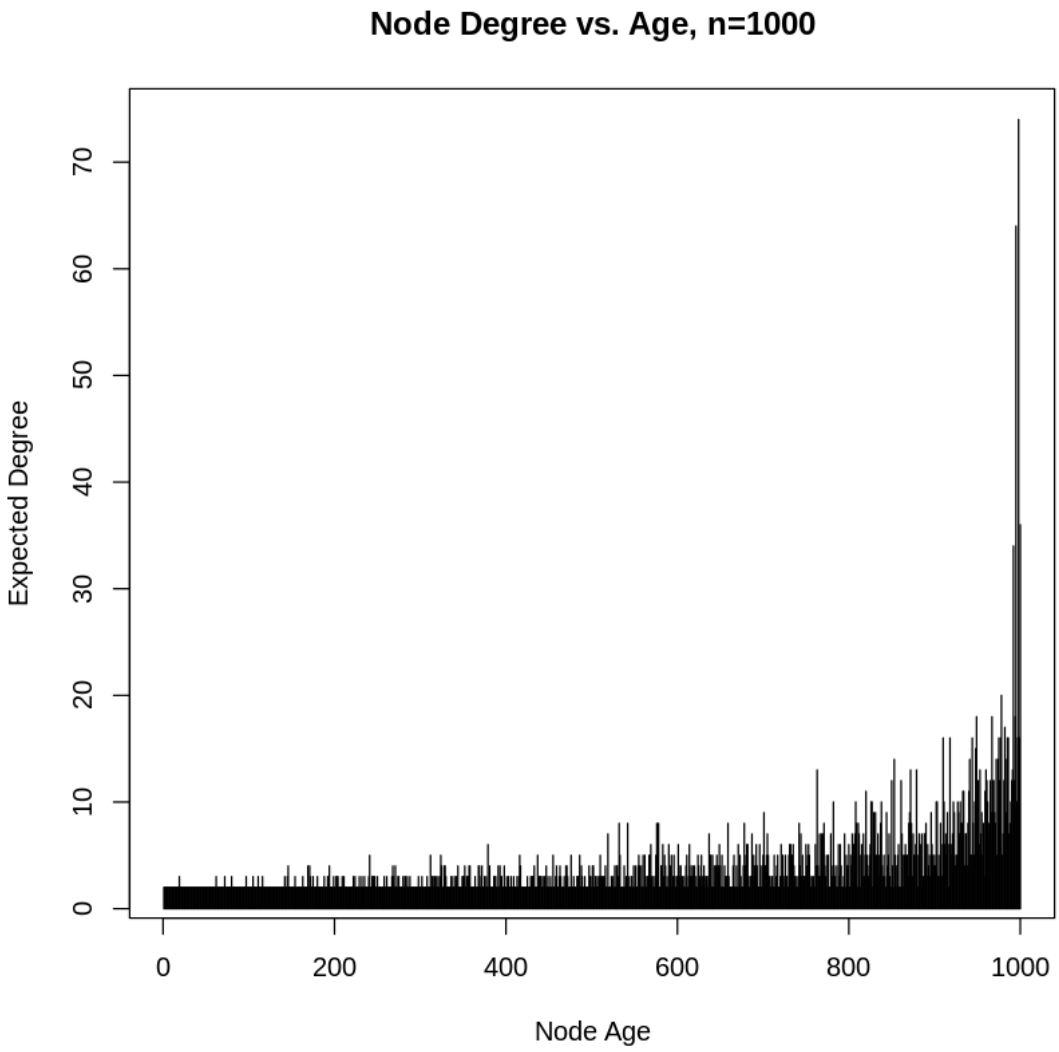


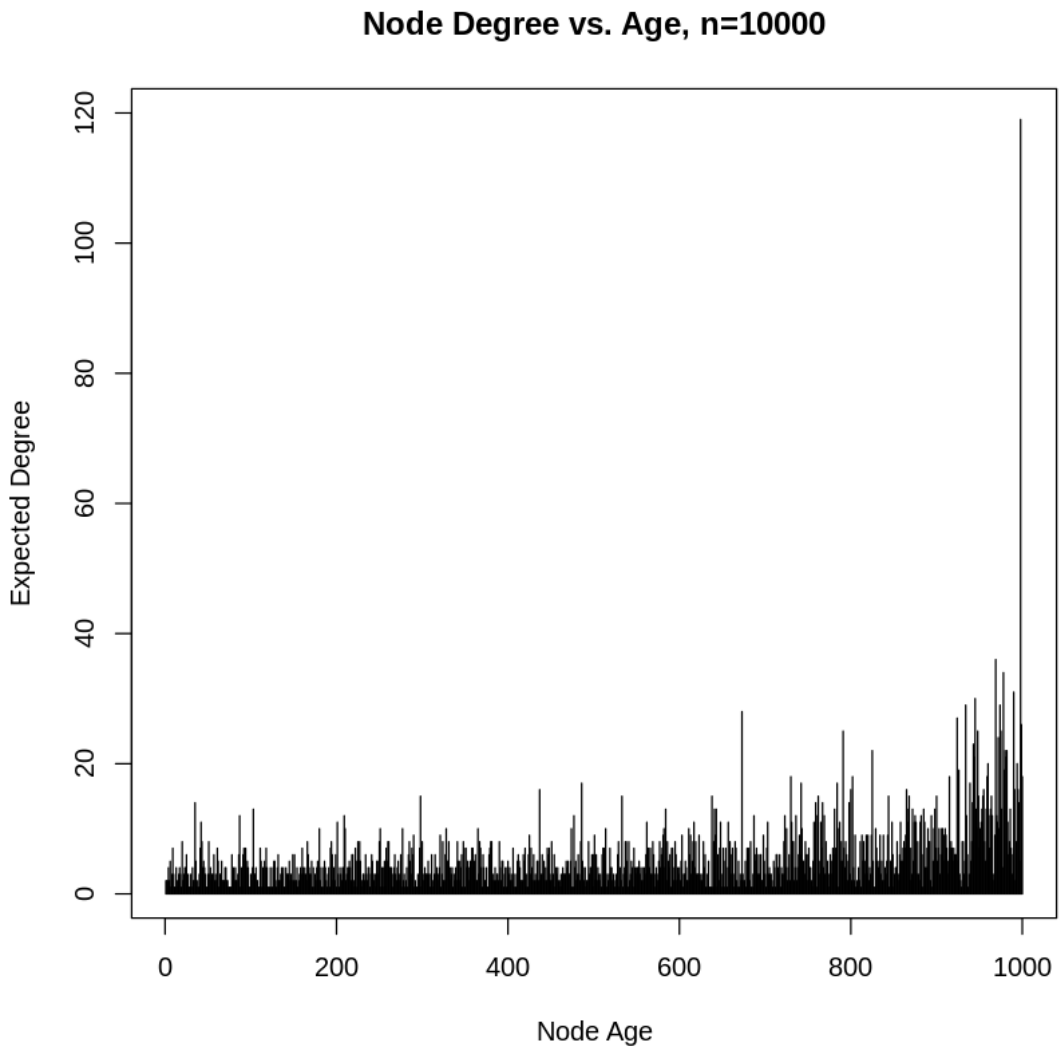
node_neighbor_degree: -0.850264769732953



node_neighbor_degree: -1.52665015366346


Degree Distribution, n=10000
log(frequency)
2
3
4
5
6
7
0
1
2
3
4
log(degree)


Node Degree vs. Age, n=1000
Expected Degree
0
10
20
30
40
50
60
70
0
200
400
600
800
1000
Node Age



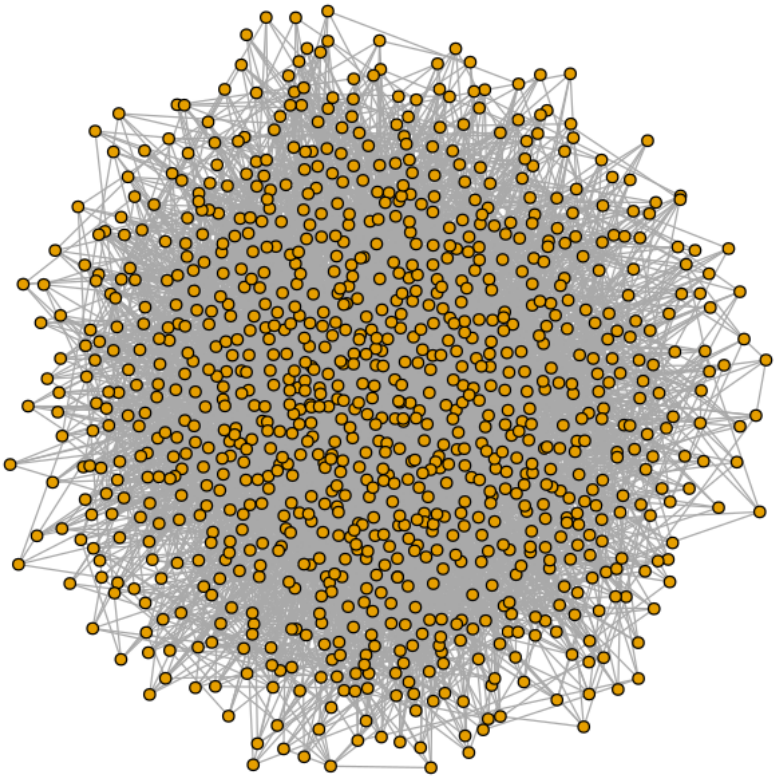
m=5

[ ]:
```
counter = 0
for (iteration in 1:500){
g1 <- sample_pa(n=1000, m=5, directed = FALSE)
if (is.connected(g1)){
  counter = counter + 1
  }
}
print(sprintf("Connected Probability: %s", counter/500))
plot(g1, vertex.size=3, vertex.label=NA)
community_1 <- cluster_fast_greedy(g1)
comm_modularity_1 <- modularity(g1, membership(community_1))
```

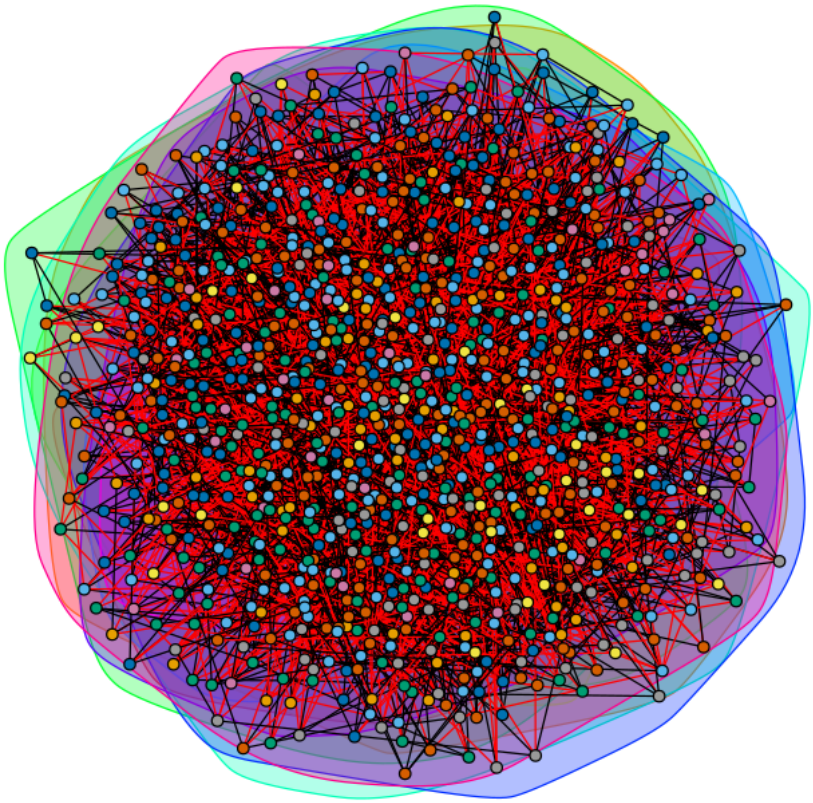
```
print(sprintf("Modularity: %s", comm_modularity_1))
plot(community_1, g1, vertex.size=3, vertex.label=NA)
counter = 0
for (iteration in 1:500){
g2 <- sample_pa(n=10000, m=1, directed = FALSE)
if (is.connected(g2)){
  counter = counter + 1
  }
}
community <- cluster_fast_greedy(g2)
comm_modularity <- modularity(g2, membership(community))
print(sprintf("Modularity: %s", comm_modularity))
plot(community, g2, vertex.size=3, vertex.label=NA)

#n=1000
degree1 <- log(c(1:length(degree.distribution(g1))))
dist1 <- log(degree.distribution(g1))
df1 <- data.frame(x=degree1, y=dist1)
plot(df1$x, df1$y, main='Degree Distribution, n=1000', xlab = "log(degree)",␣
 ↪ylab = "log(frequency)", type='o')
df1_new <- df1
df1_new[is.na(df1_new) | df1_new == "Inf"] <- NA
df1_new[is.na(df1_new) | df1_new == "-Inf"] <- NA
lm1 = lm(y~x, data = df1_new)
coef(lm1)[2]

#n=10,000
degree2 <- log(c(1:length(degree.distribution(g2))))
dist2 <- log(degree.distribution(g2))
df2 <- data.frame(x=degree2, y=dist2)
plot(df2$x, df2$y, main='Degree Distribution, n=10000', xlab = "log(degree)",␣
 ↪ylab = "log(frequency)", type='o')
df2_new <- df2
df2_new[is.na(df2_new) | df2_new == "Inf"] <- NA
df2_new[is.na(df2_new) | df2_new == "-Inf"] <- NA
lm2 = lm(y~x, data = df2_new)
coef(lm2)[2]

#n=1000
node_neighbor_degree = c()
for (i in 1:vcount(g1)){
  node = sample(vcount(g1), 1)
  node_neighbors = neighbors(g1, node)
  if (length(node_neighbors) != 0){
    node_neighbor = sample(length(node_neighbors), 1)
    node_neighbor_degree = c(node_neighbor_degree, degree(g1,␣
 ↪node_neighbors[node_neighbor]))
```

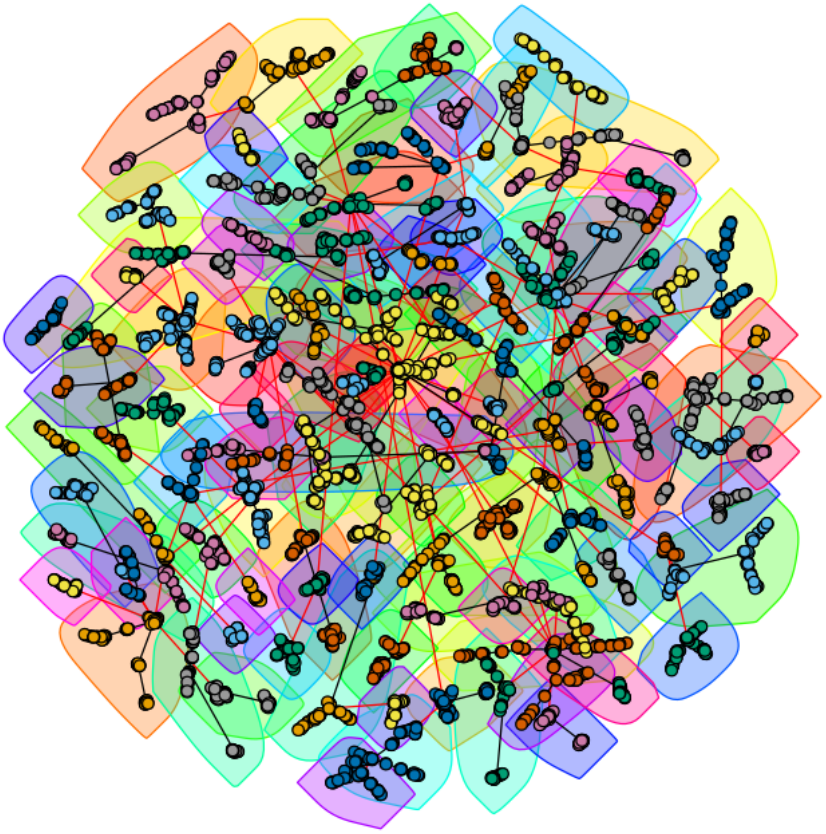
```
    }
}
df3 <- data.frame(table(node_neighbor_degree))
df3[,1] <- log(as.numeric(as.character( df3[, 1] )))
df3[,2] <- log(as.numeric(as.character( df3[, 2] )))
plot(df3$node_neighbor_degree, df3$Freq, main='Degree Distribution, n=1000',␣
 ↪xlab = "log(degree)", ylab = "log(frequency)", type='o')
df3_new <- df3
df3_new[is.na(df3_new) | df3_new == "Inf"] <- NA
df3_new[is.na(df3_new) | df3_new == "-Inf"] <- NA
lm3 = lm(Freq~node_neighbor_degree, data = df3_new)
coef(lm3)[2]

#n=10000
node_neighbor_degree = c()
for (i in 1:vcount(g2)){
  node = sample(vcount(g2), 1)
  node_neighbors = neighbors(g2, node)
  if (length(node_neighbors) != 0){
    node_neighbor = sample(length(node_neighbors), 1)
    node_neighbor_degree = c(node_neighbor_degree, degree(g2,␣
 ↪node_neighbors[node_neighbor]))
    }
}
df4 <- data.frame(table(node_neighbor_degree))
df4[,1] <- log(as.numeric(as.character( df4[, 1] )))
df4[,2] <- log(as.numeric(as.character( df4[, 2] )))
plot(df4$node_neighbor_degree, df4$Freq, main='Degree Distribution, n=10000',␣
 ↪xlab = "log(degree)", ylab = "log(frequency)", type='o')
df4_new <- df4
df4_new[is.na(df4_new) | df4_new == "Inf"] <- NA
df4_new[is.na(df4_new) | df4_new == "-Inf"] <- NA
lm4 = lm(Freq~node_neighbor_degree, data = df4_new)
coef(lm4)[2]

node_degree <- degree(g1)[1:1000]
node_age <- rev(node_degree)
plot(node_age, main="Node Degree vs. Age, n=1000",xlab="Node␣
 ↪Age",ylab="Expected Degree", type='h')
node_degree <- degree(g2)[1:1000]
node_age <- rev(node_degree)
plot(node_age, main="Node Degree vs. Age, n=10000",xlab="Node␣
 ↪Age",ylab="Expected Degree", type='h')
```

```
[1] "Connected Probability: 1"
[1] "Modularity: 0.28111630780003"
```

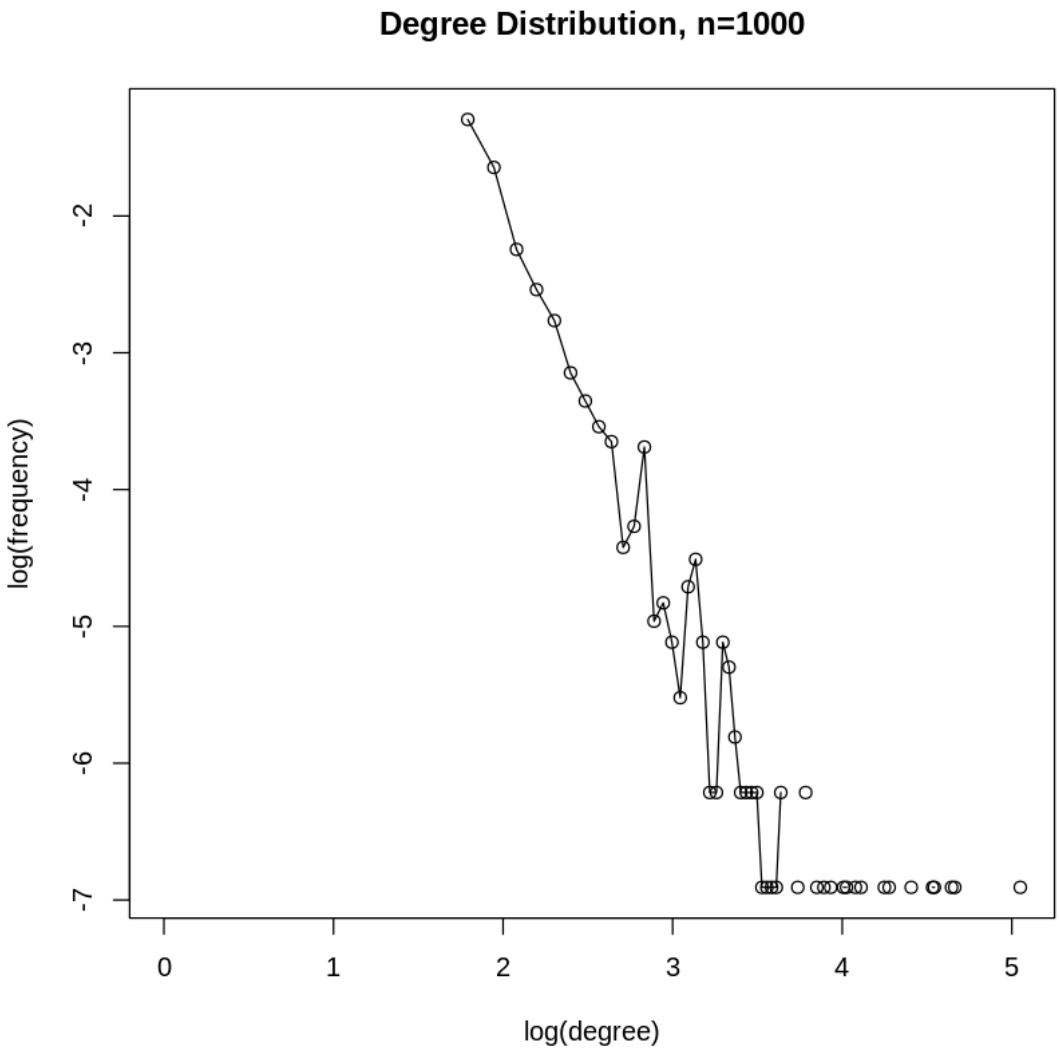


```
[1] "Modularity: 0.978492378690815"
```

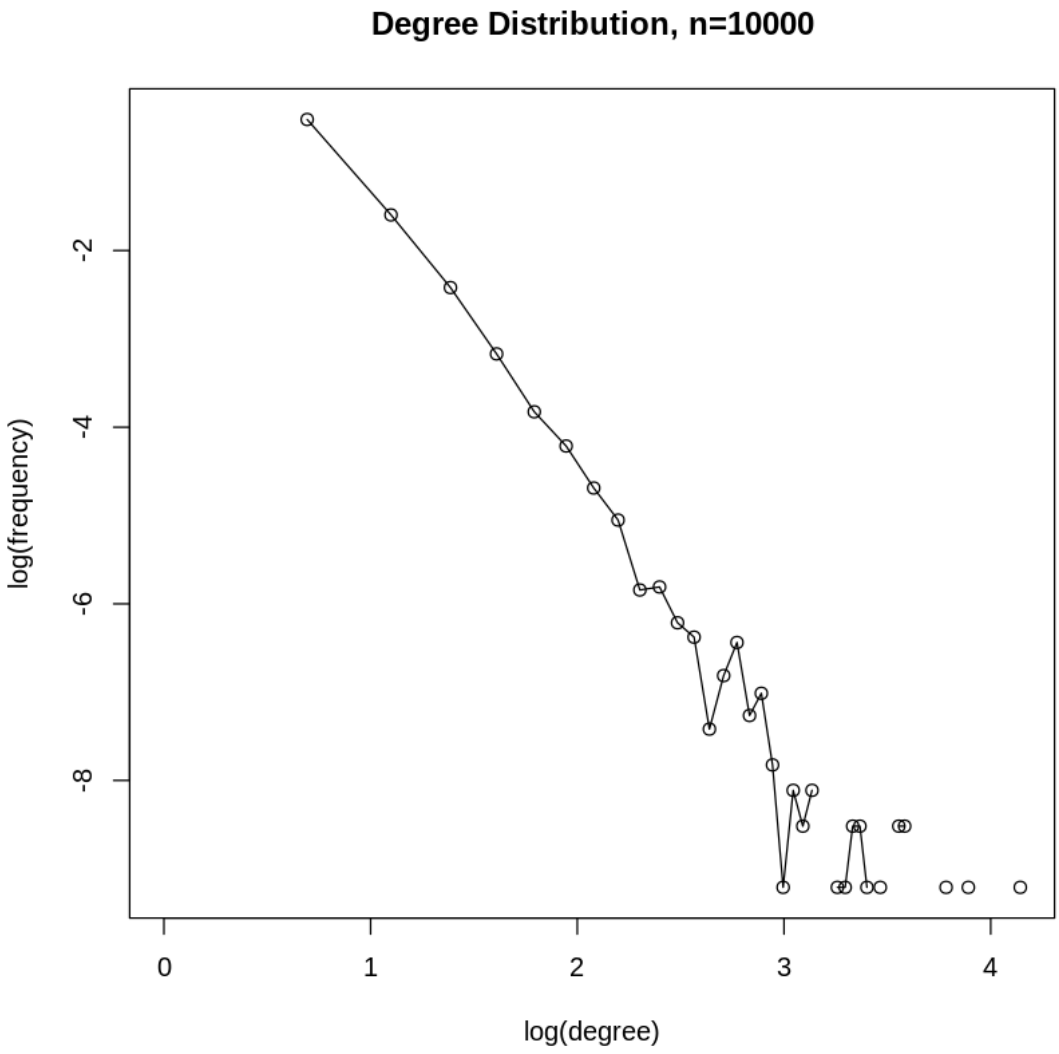




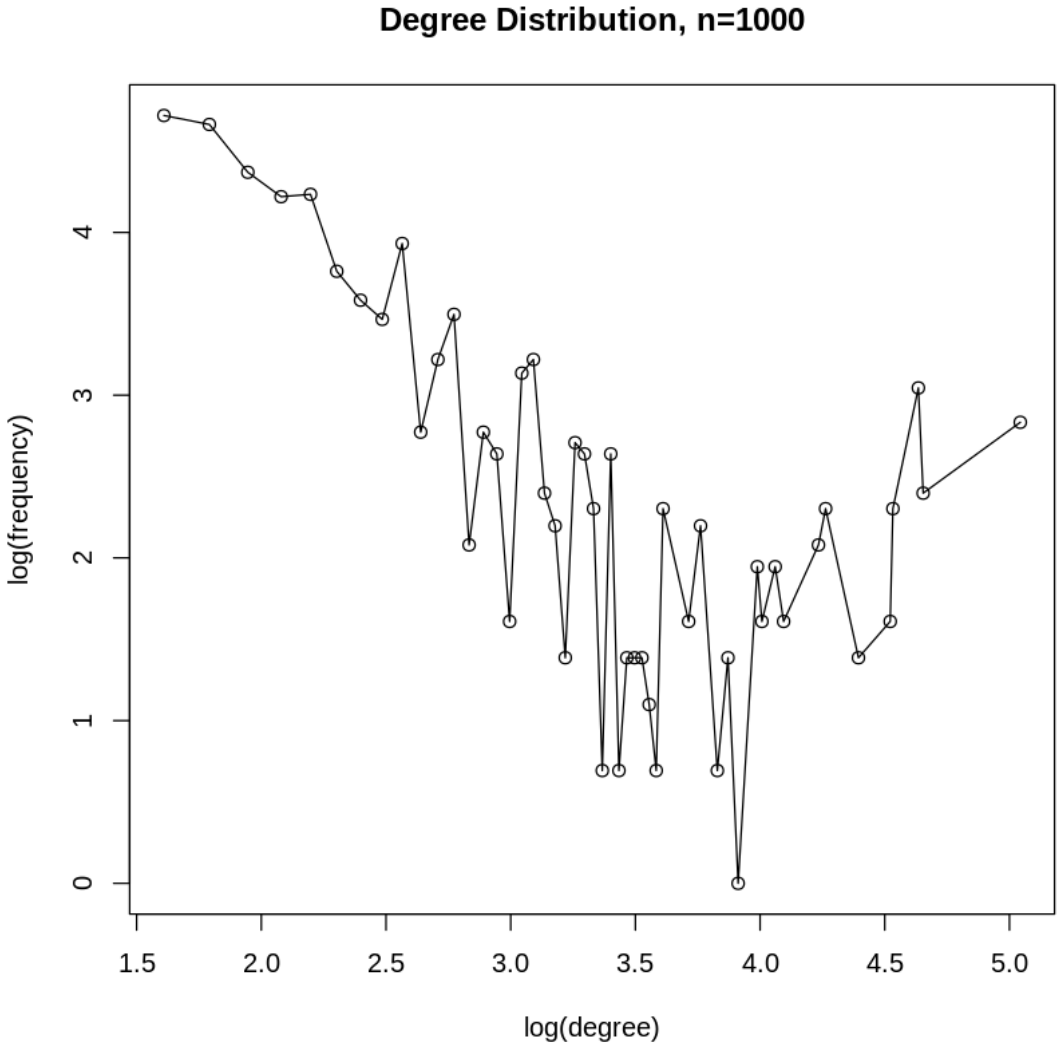
x: -1.94123554478617



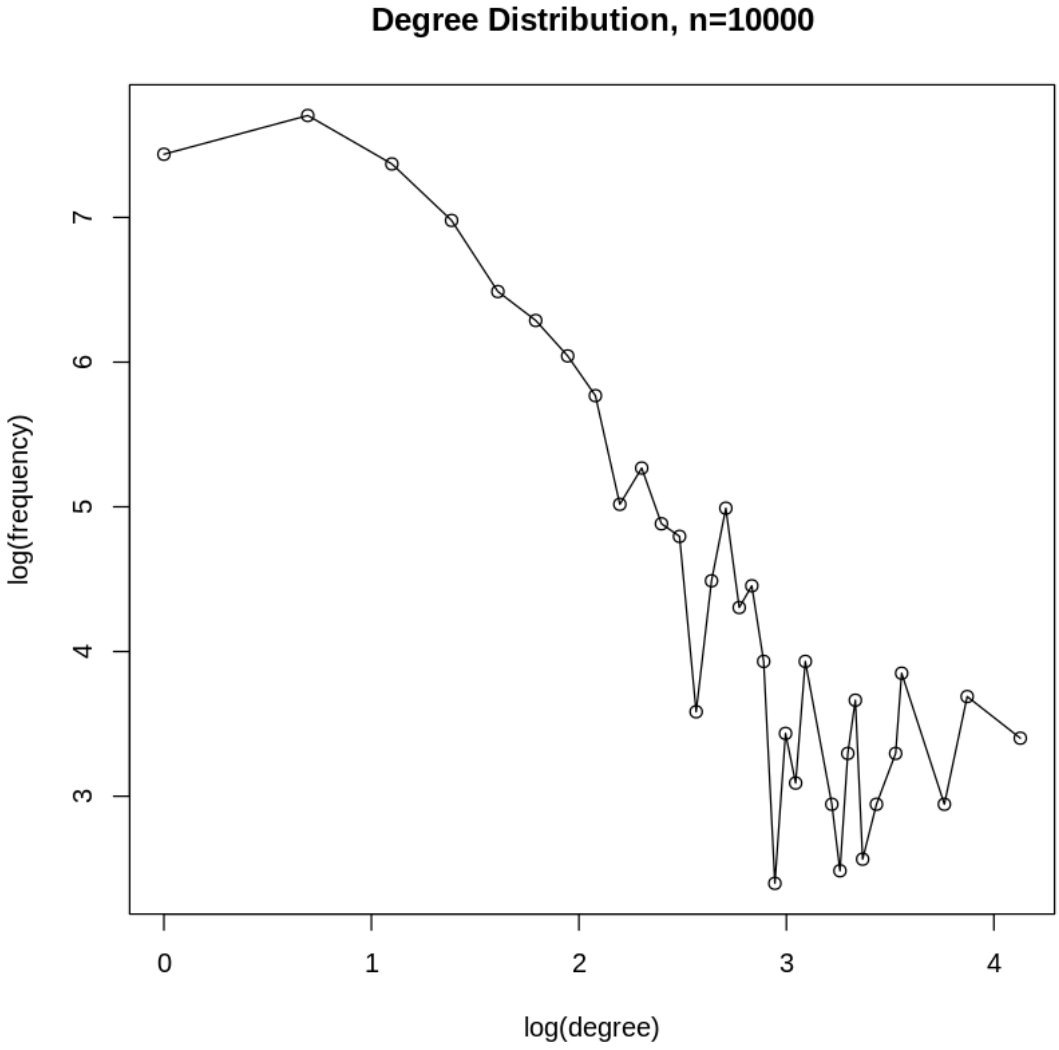
x: -2.89850434410001

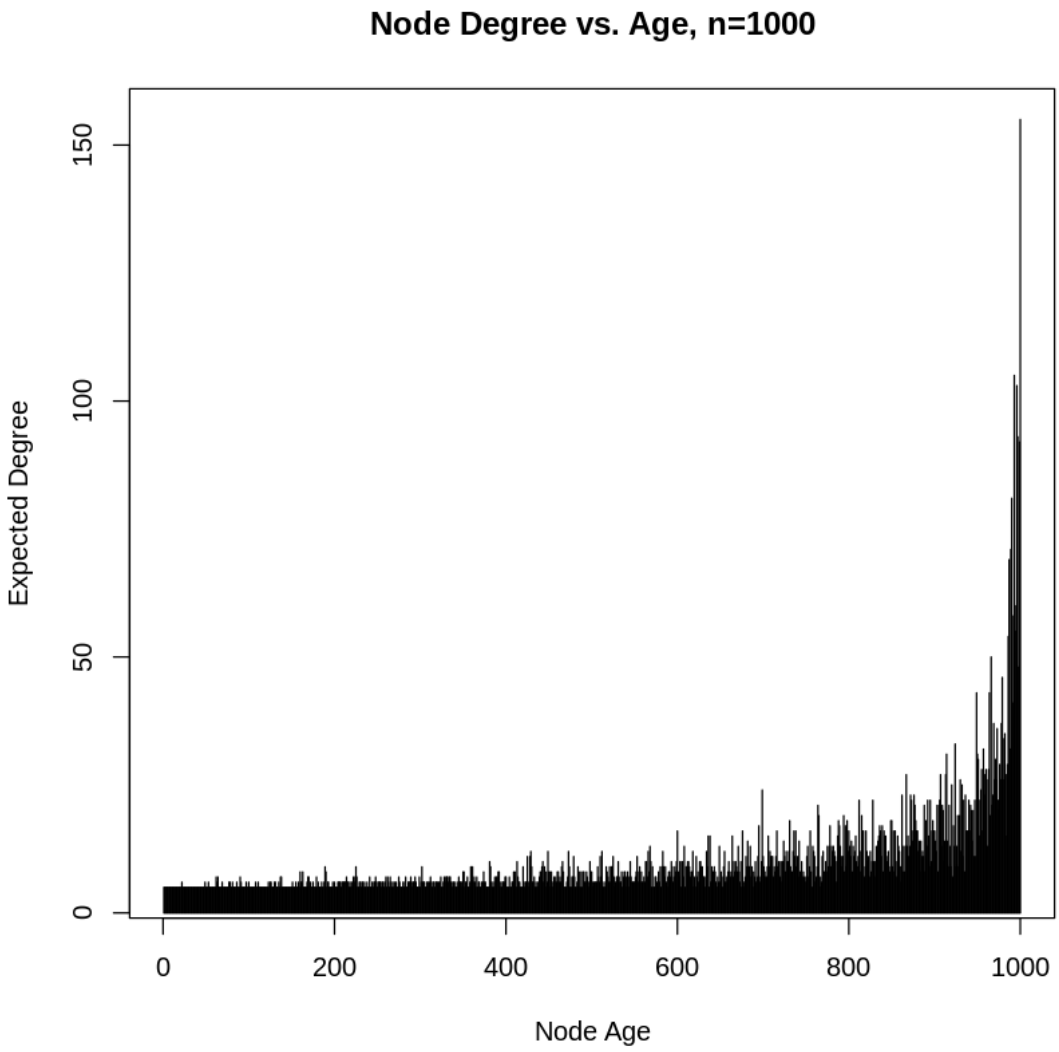


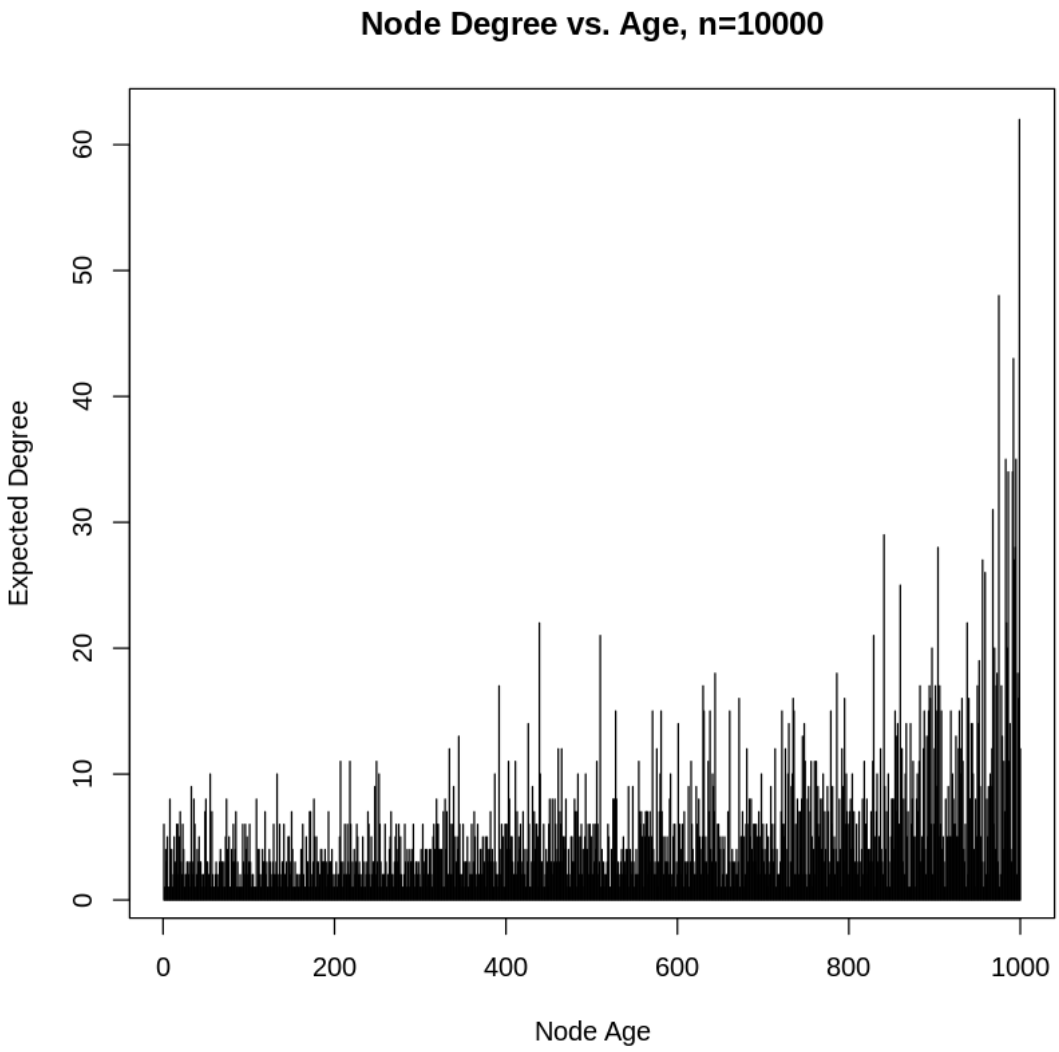
node_neighbor_degree: -0.903138404295475



node_neighbor_degree: -1.49754955573079


Degree Distribution, n=10000
log(frequency)
3
4
5
6
7
0
1
2
3
4
log(degree)


Node Degree vs. Age, n=1000
Expected Degree
0
50
100
150
0
200
400
600
800
1000
Node Age



As the number of edges added in each step increases, the following can be observed:

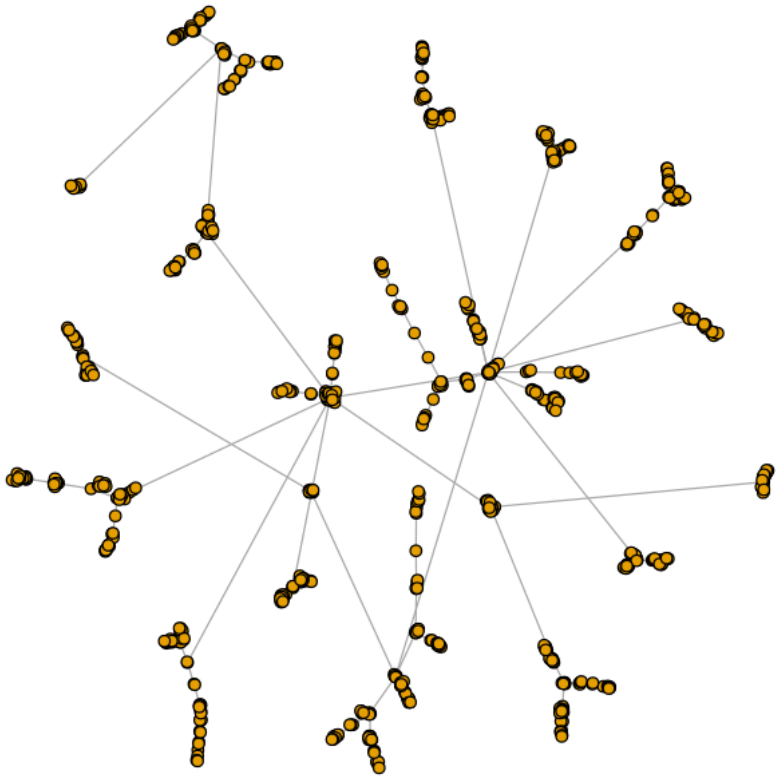
- Expected degree increases as the value of "m" increases but not much difference is observed in the node degree vs. age distribution.
- The slope of the degree distribution graph becomes slightly more negative as m increases.
- Modularity for n=1000 decreases as m increases.

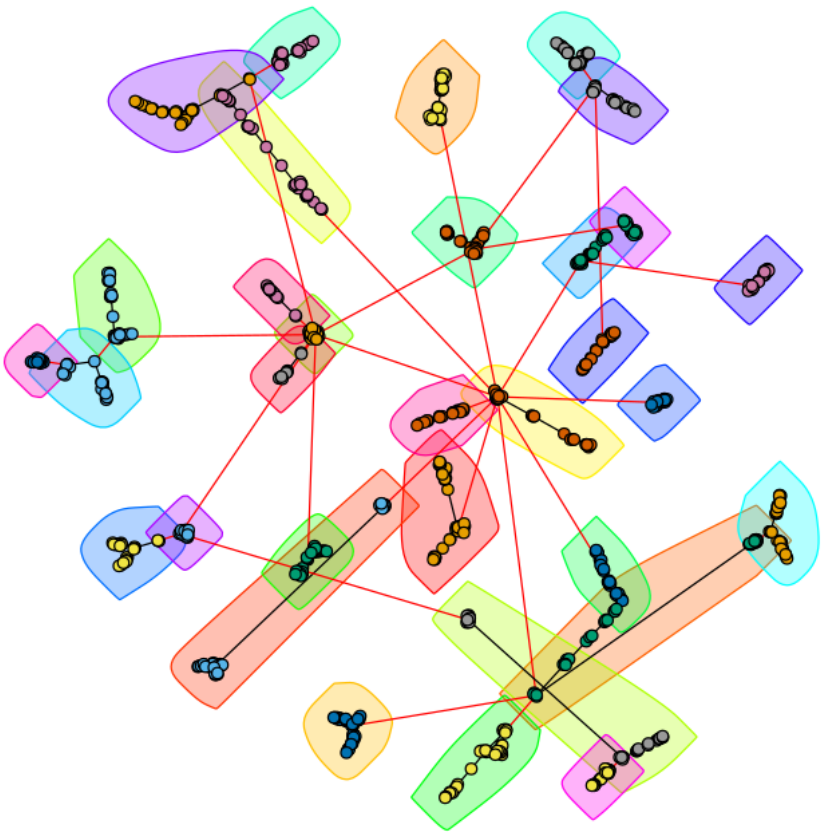
14 2(h)

[ ]:
```
#original
counter = 0
for (iteration in 1:500){
g <- sample_pa(n=1000, m=1, directed = FALSE)
```

```
if (is.connected(g)){
  counter = counter + 1
  }
}

community_g <- cluster_fast_greedy(g)
comm_modularity_g <- modularity(g, membership(community_g))
print(sprintf("Modularity: %s", comm_modularity_g))

plot(g, vertex.size=3, vertex.label=NA)
plot(community_g, g, vertex.size=3, vertex.label=NA)
```

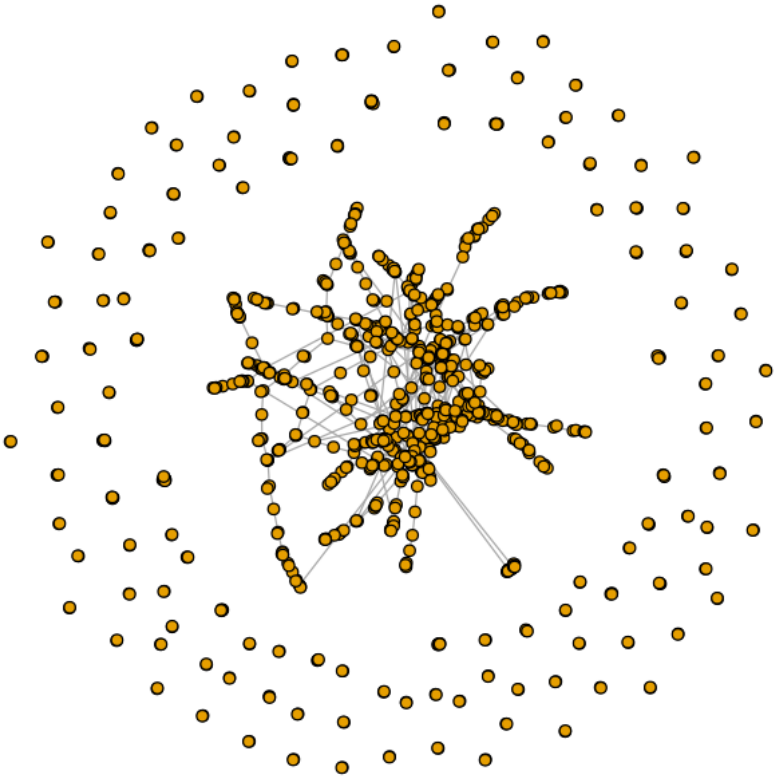
```
[1] "Modularity: 0.934463993523053"
```

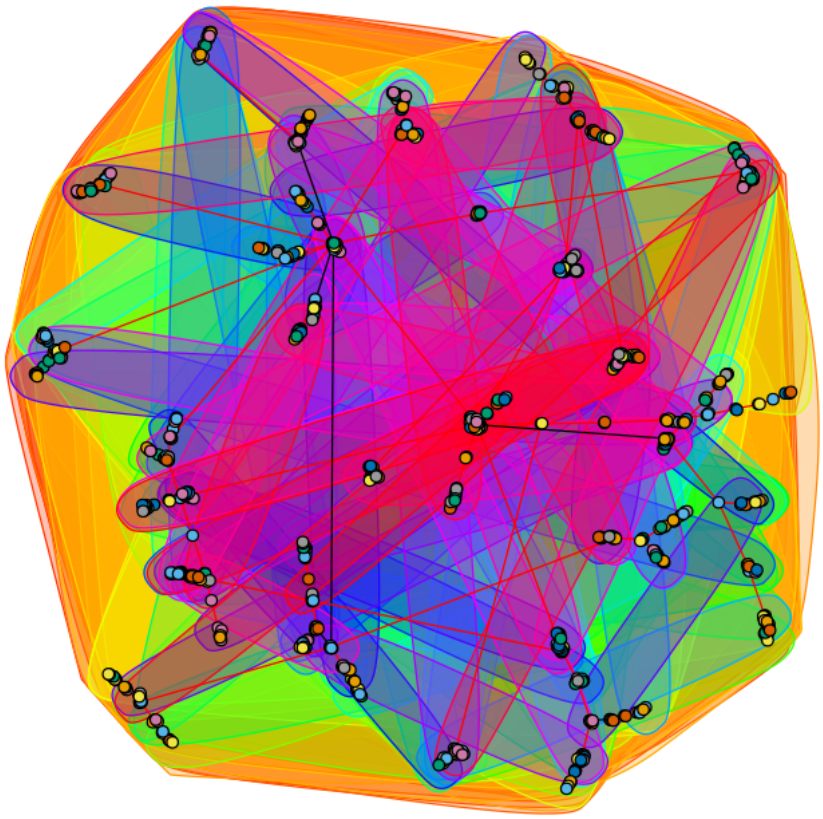




[ ]:
```
#matched
g_matched = sample_degseq(degree(g), method="simple.no.multiple")

community_g_matched <- cluster_fast_greedy(g_matched)
comm_modularity_g_matched <- modularity(g_matched,␣
 ↪membership(community_g_matched))
print(sprintf("Modularity: %s", comm_modularity_g_matched))

plot(g_matched, vertex.size=3, vertex.label=NA)
plot(community_g_matched, g, vertex.size=3, vertex.label=NA)
```

```
[1] "Modularity: 0.844995145295442"
```



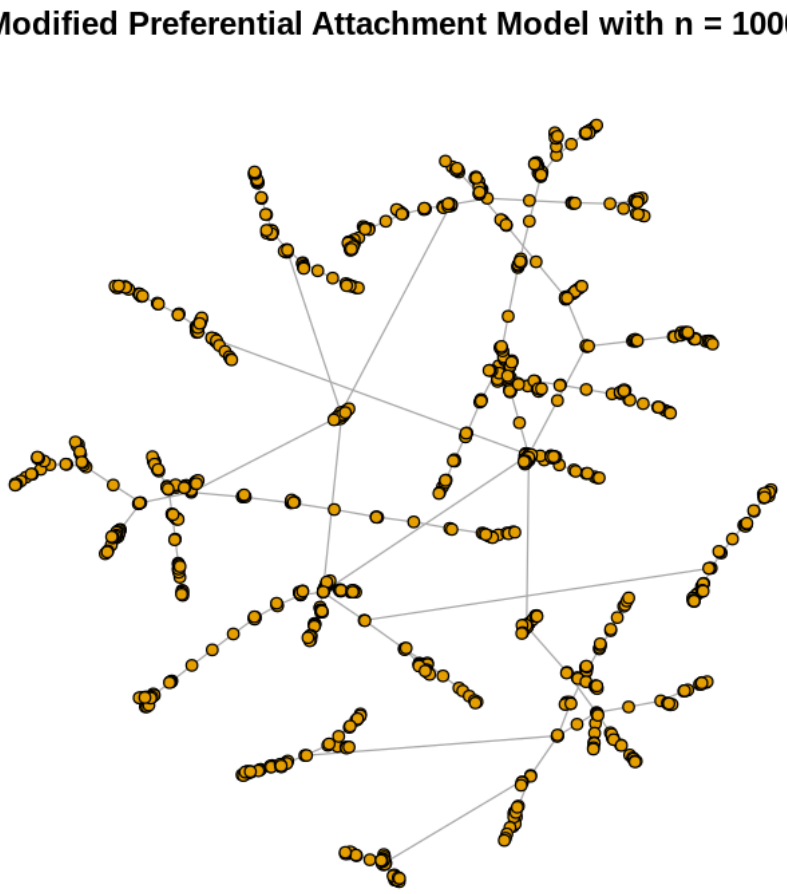


The modularity is lower with a matched network created using the stub-matching procedure because as one may observe in the graph, there are disconnected nodes.

15 3(a)

[ ]:
```
g <- sample_pa_age(n=1000, directed=FALSE, m=1, pa.exp=1, aging.exp=-1, zero.
↪deg.appeal=1, zero.age.appeal=0, deg.coef=1, age.coef=1)
```

[ ]:
```
plot(g,vertex.label="", vertex.size=3, main = "Modified Preferential Attachment␣
↪Model with n = 1000")
```

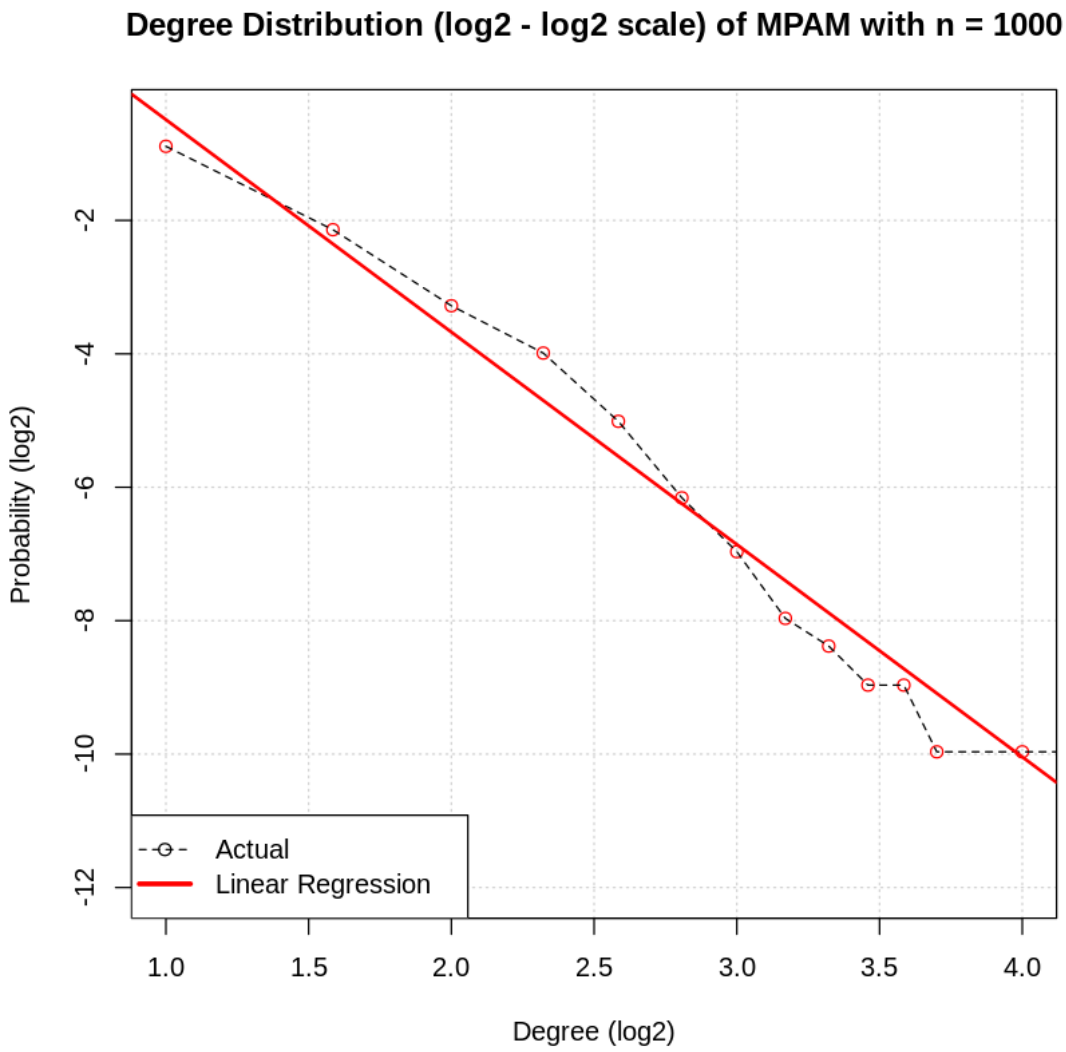


[ ]:
```
deg_dist_1 = degree.distribution(g)
deg_1 <- log2(c(1:length(deg_dist_1)))[which(deg_dist_1 != 0, arr.ind = TRUE)]
dist_1 <- log2(deg_dist_1)[which(deg_dist_1 != 0, arr.ind = TRUE)]
```

[ ]:
```
print("Slope and Intercept for MPAM:")
print(lm(dist_1 ~ deg_1))

plot(deg_1, dist_1, main="Degree Distribution (log2 - log2 scale) of MPAM with␣
 ↪n = 1000", xlab="Degree (log2)", ylab="Probability (log2)", grid(),␣
 ↪col="red", ylim=c(-12, -0.5), xlim=c(1, 4))
lines(deg_1, dist_1, lty=2)
abline(lm(dist_1 ~ deg_1), lwd=2, col="red")
```

```
legend('bottomleft', legend=c("Actual", "Linear Regression"), lty=c(2, 1),␣
↪lwd=c(1, 3), pch=c(1, NA), col=c('black', 'red'))
```

```
[1] "Slope and Intercept for MPAM:"

Call:
lm(formula = dist_1 ~ deg_1)

Coefficients:
(Intercept)        deg_1
      2.695       -3.184
```



The slope of the linear regression line with n = 1000 is -3.184. Therefore, the power law exponent is 3.184. As expected for preferential attachment models, it is about equal to 3.

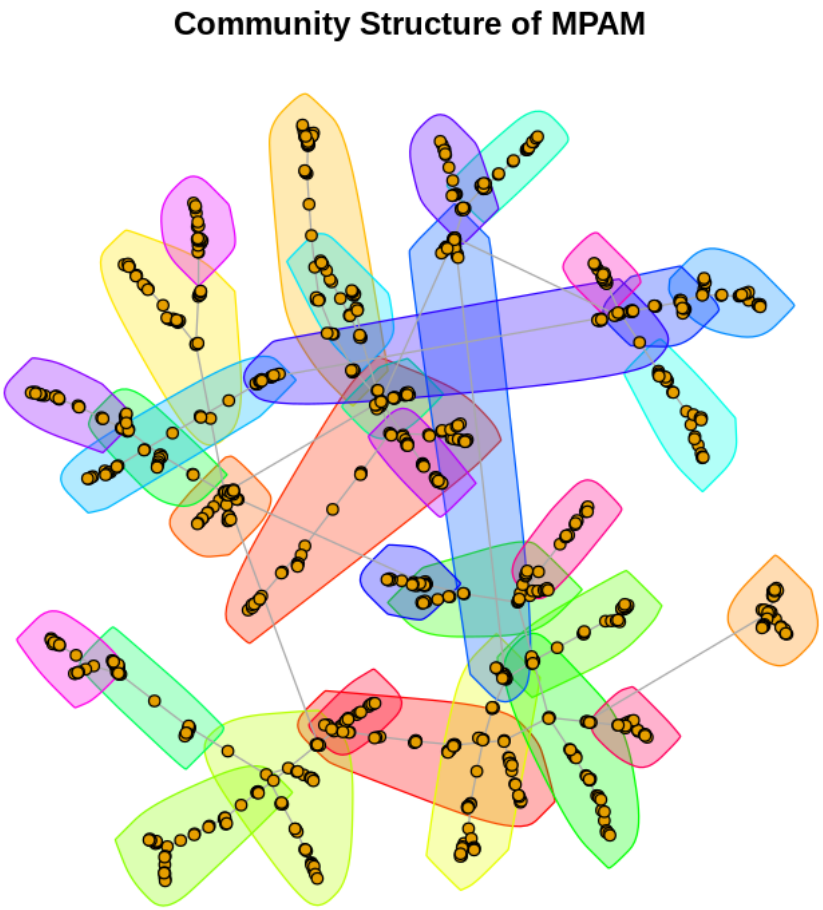
16 3(b)

[ ]:
```
comm_struct <- cluster_fast_greedy(g)
mod <- modularity(comm_struct)
```

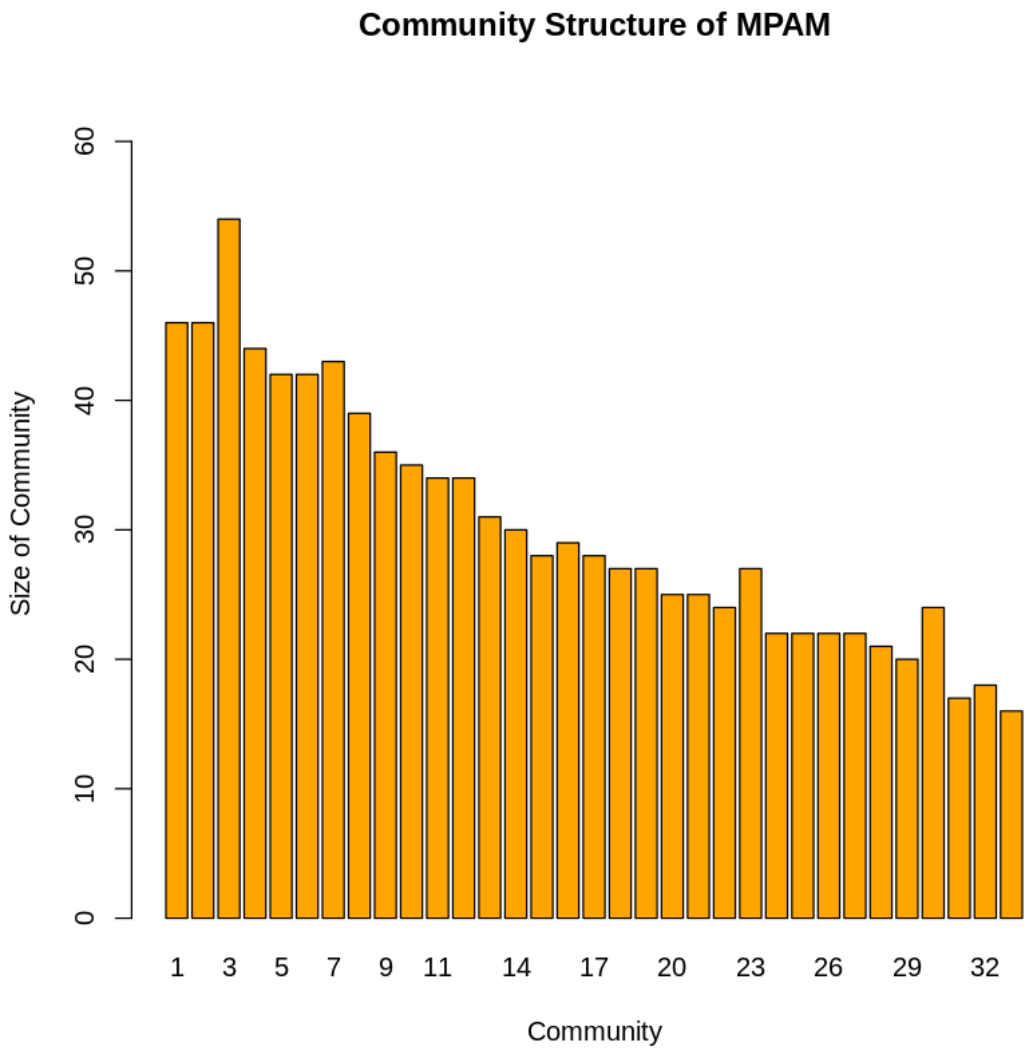
[ ]:
```
print(sprintf("Modularity: %f", mod))
```

```
[1] "Modularity: 0.934524"
```

[ ]:
```
plot(g, mark.groups=groups(comm_struct), vertex.size=3, vertex.label="",␣
↪main="Community Structure of MPAM")
```



```
[ ]: barplot(as.vector(sizes(comm_struct)), names.arg=seq(1, length(comm_struct),␣
      ↪1), main="Community Structure of MPAM",
       xlab="Community", ylab="Size of Community", ylim=c(0, max(as.
      ↪vector(sizes(comm_struct)))+10), col="orange")
```

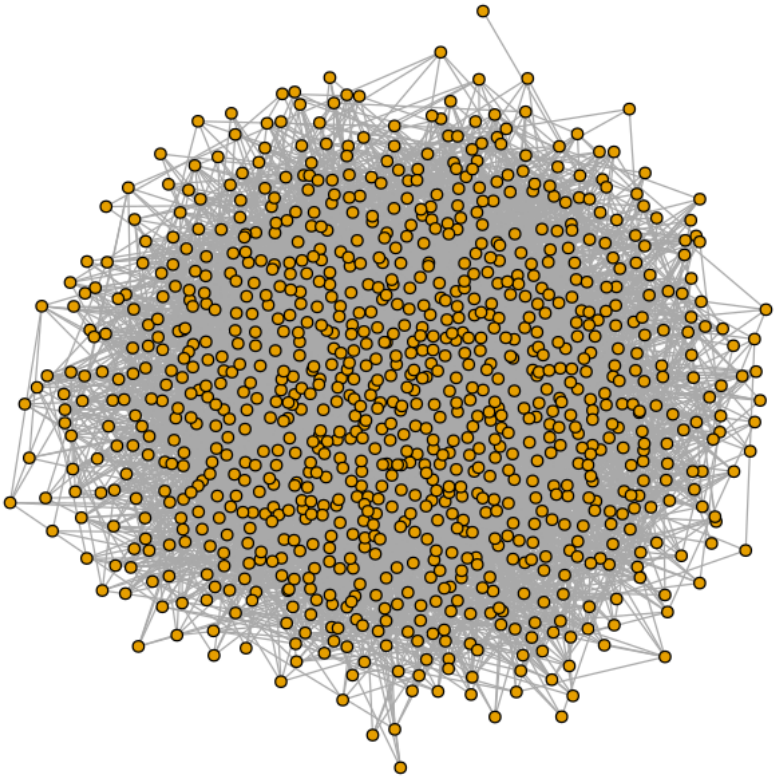


17 Part2

18 1(a)

[ ]:
```
g <- erdos.renyi.game(n=1000, type="gnp", p=0.01, directed=FALSE)
```

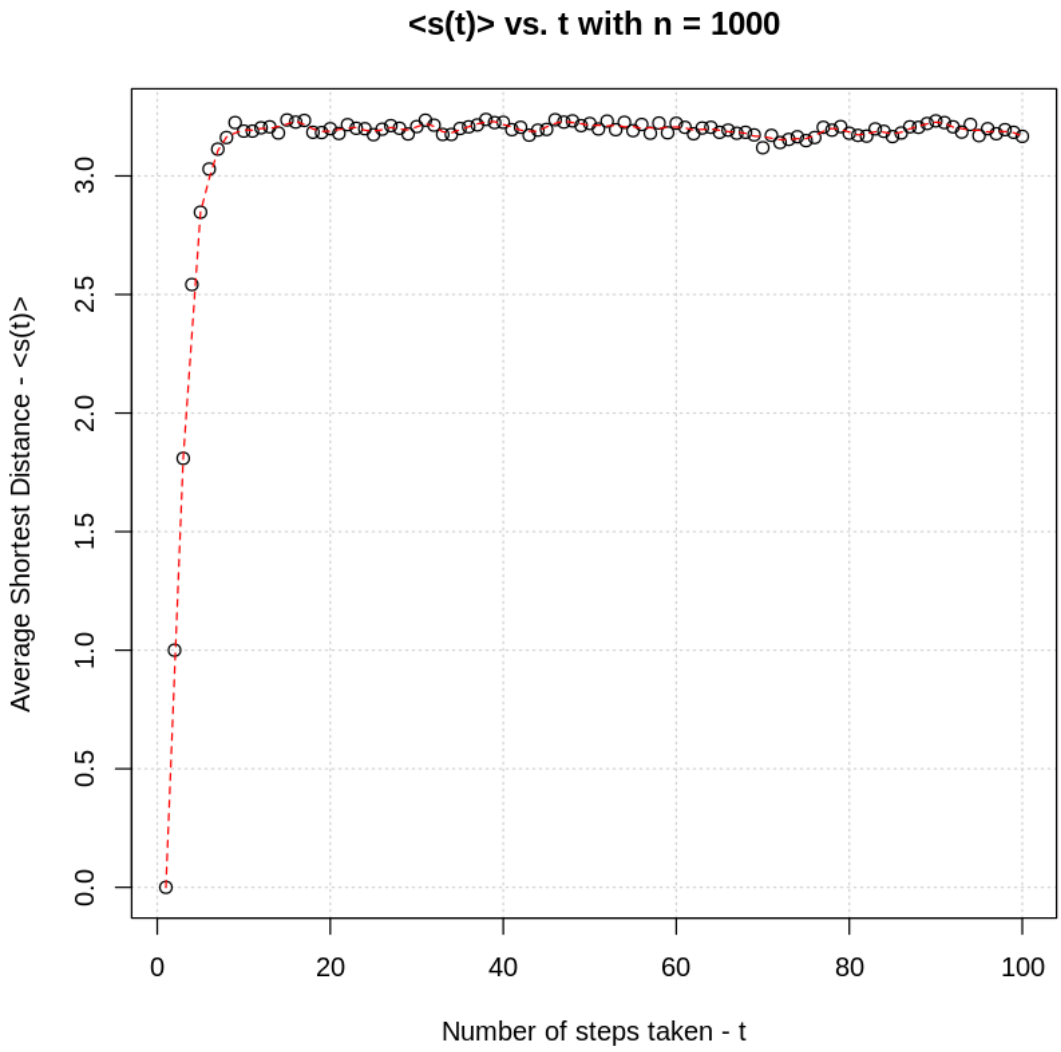
[ ]:
```
plot(g, vertex.label="", vertex.size=3, main="Undirected Erdos Renyi Network␣
↪with n = 1000 and p = 0.01")
```



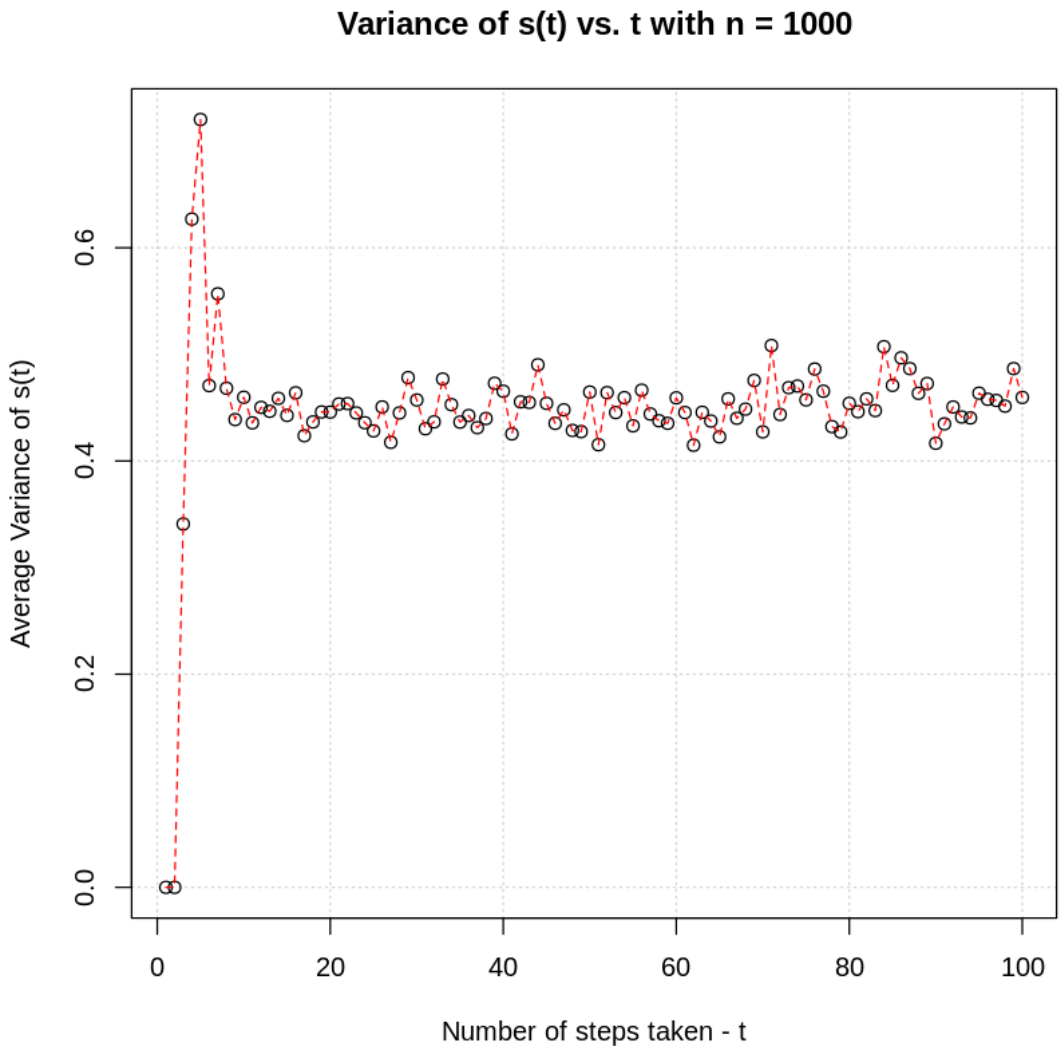
19 1(b)

[ ]:
```
s_t = matrix(data=0.0, nrow=1000, ncol=100)
term_node_deg = matrix(data=0.0, nrow=1000, ncol=1)

for (i in 1:1000) {
    if (is_connected(g)) {
        gcc <- g
    }
    else {
        g.components <- clusters(g)
        ix <- which.max(g.components$csize)
        g.giant <- induced.subgraph(g, which(g.components$membership == ix))
        gcc <- g.giant
    }
    start = sample(V(gcc), 1)
    walked_nodes = random_walk(gcc, steps=100, start)
    s_t[i,] = shortest.paths(gcc, walked_nodes, start)
    term_node_deg[i, 1] = degree(gcc, walked_nodes[length(walked_nodes)])
}
```

[ ]:
```
plot(seq(1, 100, 1), colMeans(s_t), grid(), xlab="Number of steps taken - t",␣
 ↪ylab="Average Shortest Distance - <s(t)>", main="<s(t)> vs. t with n = 1000")
lines(lowess(seq(1, 100, 1), colMeans(s_t), f=0.04), col="red", lwd=1, lty=2)
```

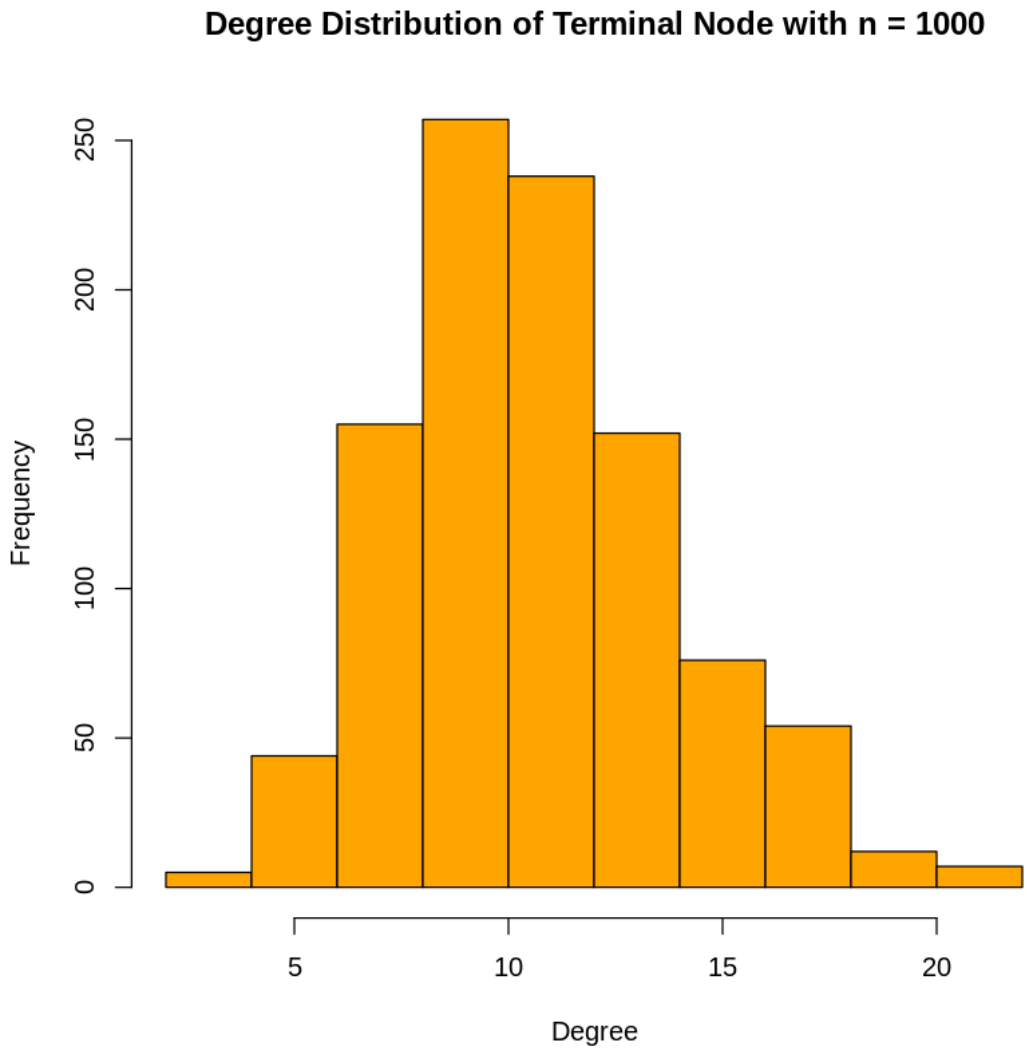


[ ]:
```
plot(seq(1, 100, 1), colVars(s_t), grid(), xlab="Number of steps taken - t",␣
  ↪ylab="Average Variance of s(t)", main="Variance of s(t) vs. t with n = 1000")
lines(lowess(seq(1, 100, 1), colVars(s_t), f=0.03), col="red", lwd=1, lty=2)
```

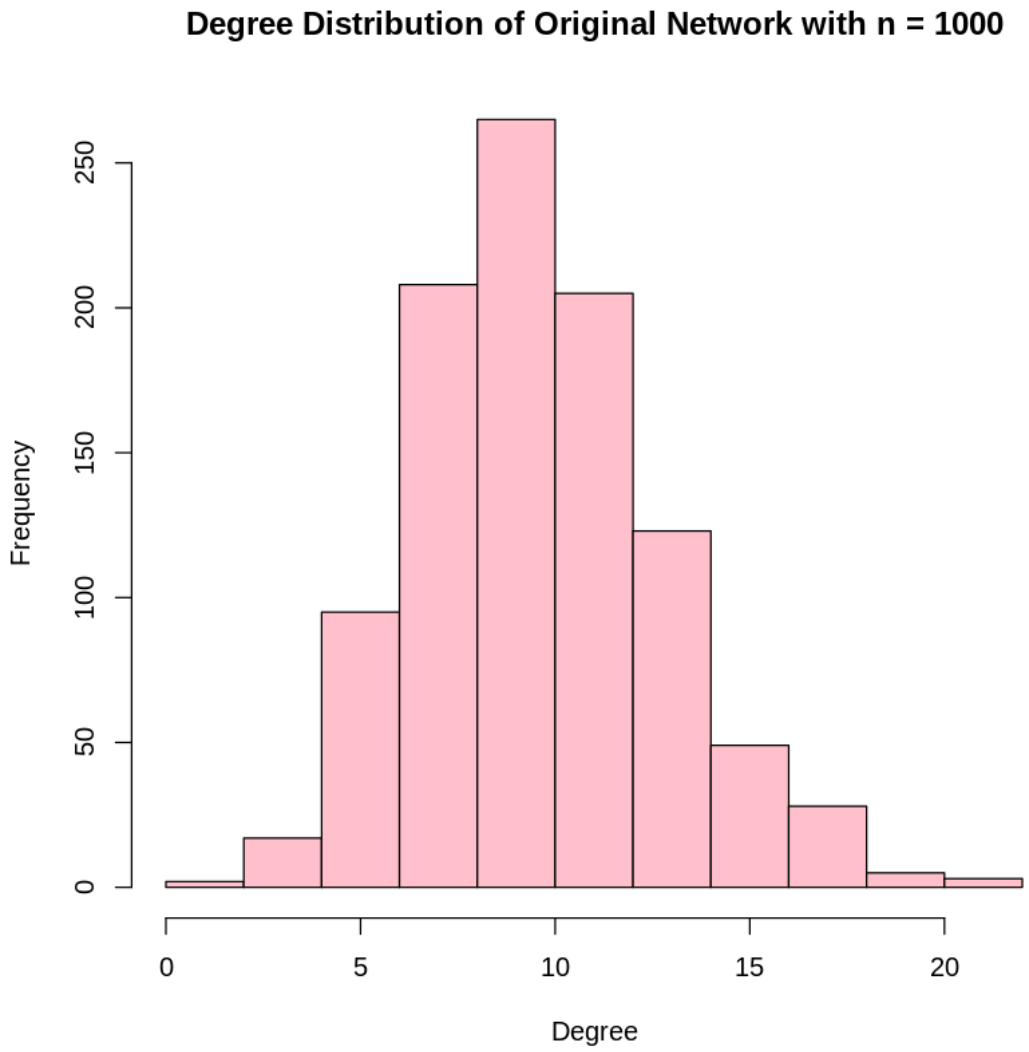


20 1(c)

[ ]:
```
hist(term_node_deg, col='orange', main="Degree Distribution of Terminal Node␣
↪with n = 1000", xlab="Degree", ylab="Frequency")
```



[ ]:
```
hist(degree(g), col="pink", main="Degree Distribution of Original Network with␣
↪n = 1000", xlab="Degree", ylab="Frequency")
```



[ ]:
```
print(diameter(g))
```

```
[1] 6
```

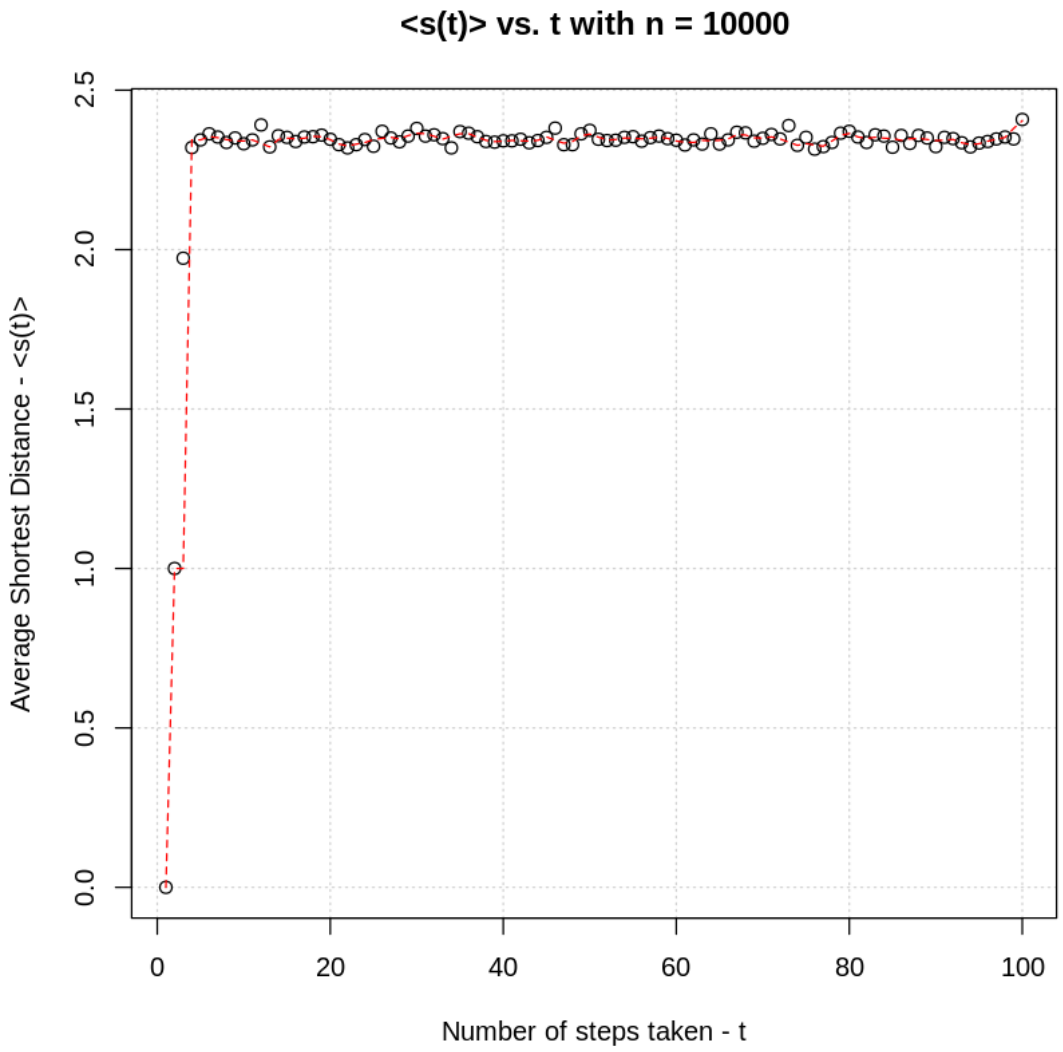
Comparing these two graphs, we can see that the degree distribution of terminal node follows the degree distribution of entire network (Binomial for Erdos-Renyi networks) with similar mean and variance.

21 1(d)

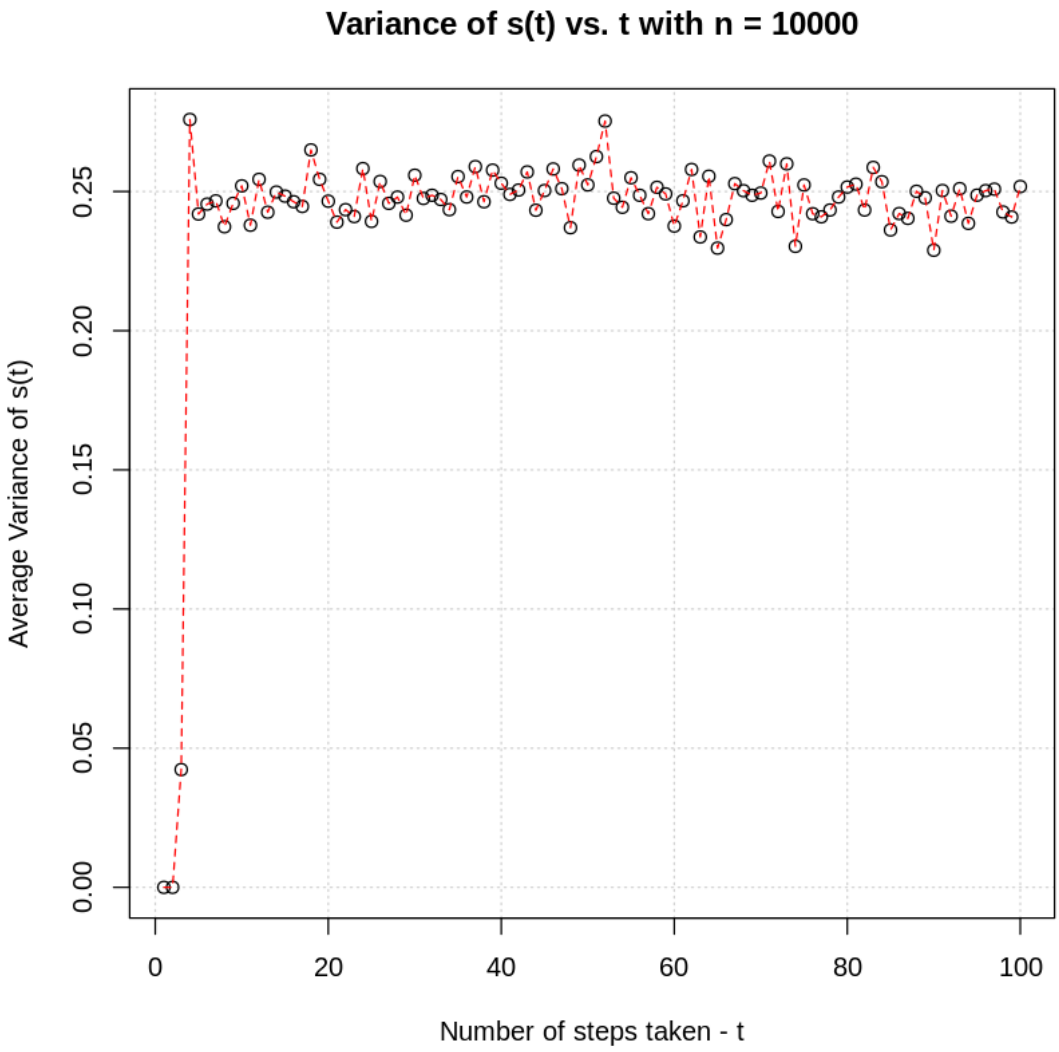
- 10000 Nodes

[ ]:
```
g <- erdos.renyi.game(n=10000, type="gnp", p=0.01, directed=FALSE)
s_t = matrix(data=0.0, nrow=1000, ncol=100)
term_node_deg = matrix(data=0.0, nrow=1000, ncol=1)

for (i in 1:1000) {
    if (is_connected(g)) {
        gcc <- g
    }
    else {
        g.components <- clusters(g)
        ix <- which.max(g.components$csize)
        g.giant <- induced.subgraph(g, which(g.components$membership == ix))
        gcc <- g.giant
    }
    start = sample(V(gcc), 1)
    walked_nodes = random_walk(gcc, steps=100, start)
    s_t[i,] = shortest.paths(gcc, walked_nodes, start)
    term_node_deg[i, 1] = degree(gcc, walked_nodes[length(walked_nodes)])
}
```

[ ]:
```
plot(seq(1, 100, 1), colMeans(s_t), grid(), xlab="Number of steps taken - t",␣
 ↪ylab="Average Shortest Distance - <s(t)>", main="<s(t)> vs. t with n =␣
 ↪10000")
lines(lowess(seq(1, 100, 1), colMeans(s_t), f=0.05), col="red", lwd=1, lty=2)
```



[ ]:
```
plot(seq(1, 100, 1), colVars(s_t), grid(), xlab="Number of steps taken - t",␣
↪ylab="Average Variance of s(t)", main="Variance of s(t) vs. t with n =␣
↪10000")
lines(lowess(seq(1, 100, 1), colVars(s_t), f=0.03), col="red", lwd=1, lty=2)
```



[ ]:
```
print(diameter(g))
```

```
[1] 3
```

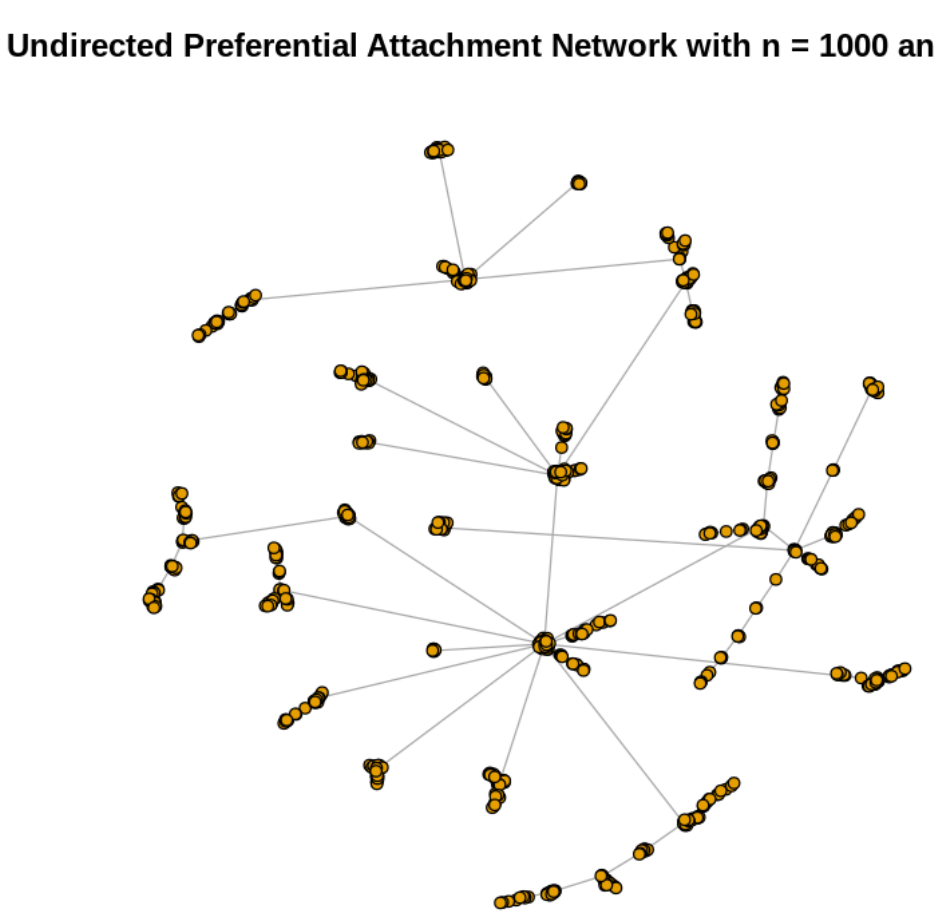
- Yes, the diameter of the network plays a role.
- The diameter of the network with 1000 nodes is 6, while the diameter of the network with 10000 nodes is 3. The diameter of larger network (10000 nodes) is shorter than the smaller network (1000 nodes).
- We can observe that the steady-state average shortest distance of the larger network is also shorter than the smaller network, 2.3 vs. 3.3.
- The number of steps required for convergence for the larger network is also smaller than the smaller network, 5 vs. 9.
- The variance of the larger network is lower than the variance of the smaller network, 0.25 vs. 0.42. It is because the average distances between nodes in the larger network are more

homogenous than the smaller network.

- These observations can be explained that a larger network has nodes tightly closed to each other. Thus, compared to a smaller network where the nodes are more spread out, the random walker covers a smaller distance for the same number of steps within a larger network.

22 2(a)

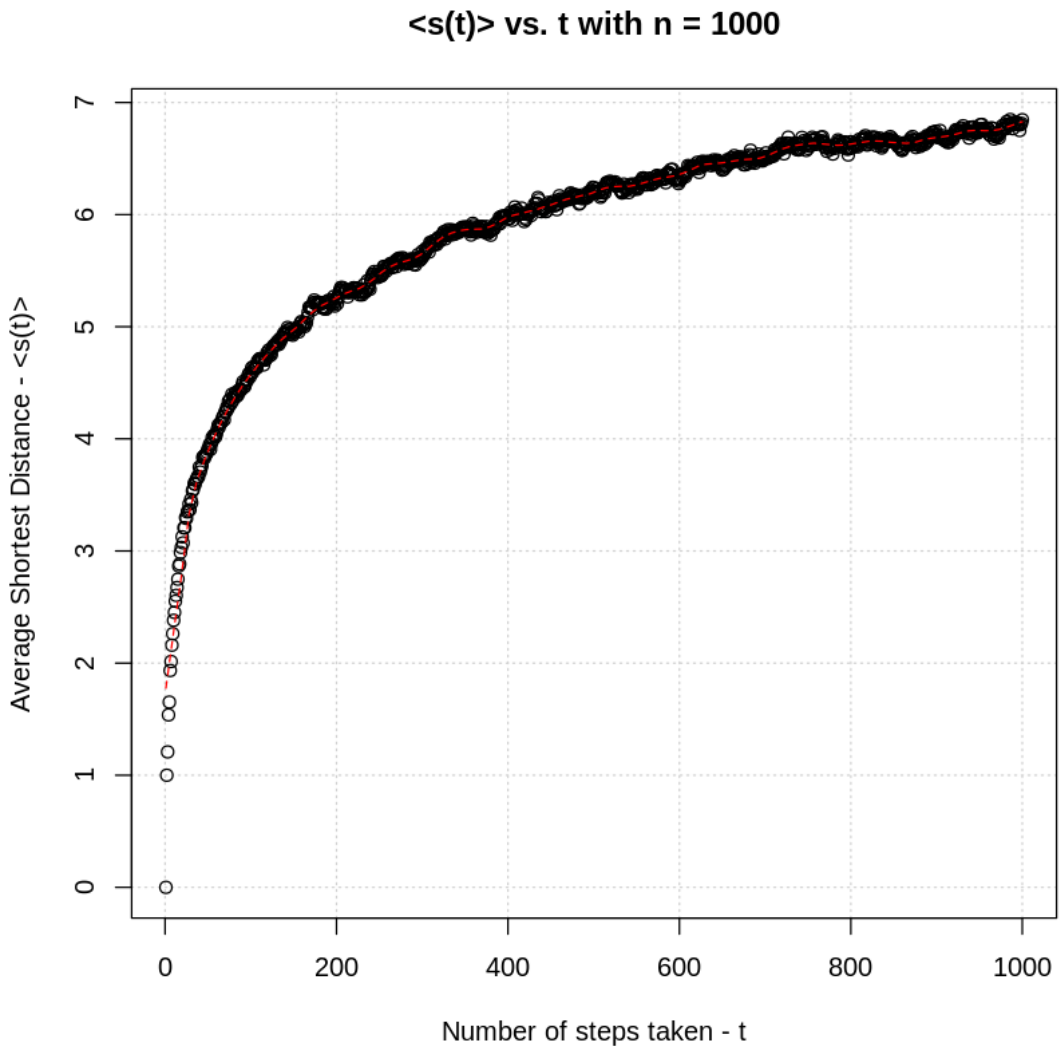
[ ]:
```
g <- barabasi.game(1000, m=1, directed=FALSE)
plot(g, vertex.label="", vertex.size=3, main="Undirected Preferential␣
  ↪Attachment Network with n = 1000 and m = 1")
```



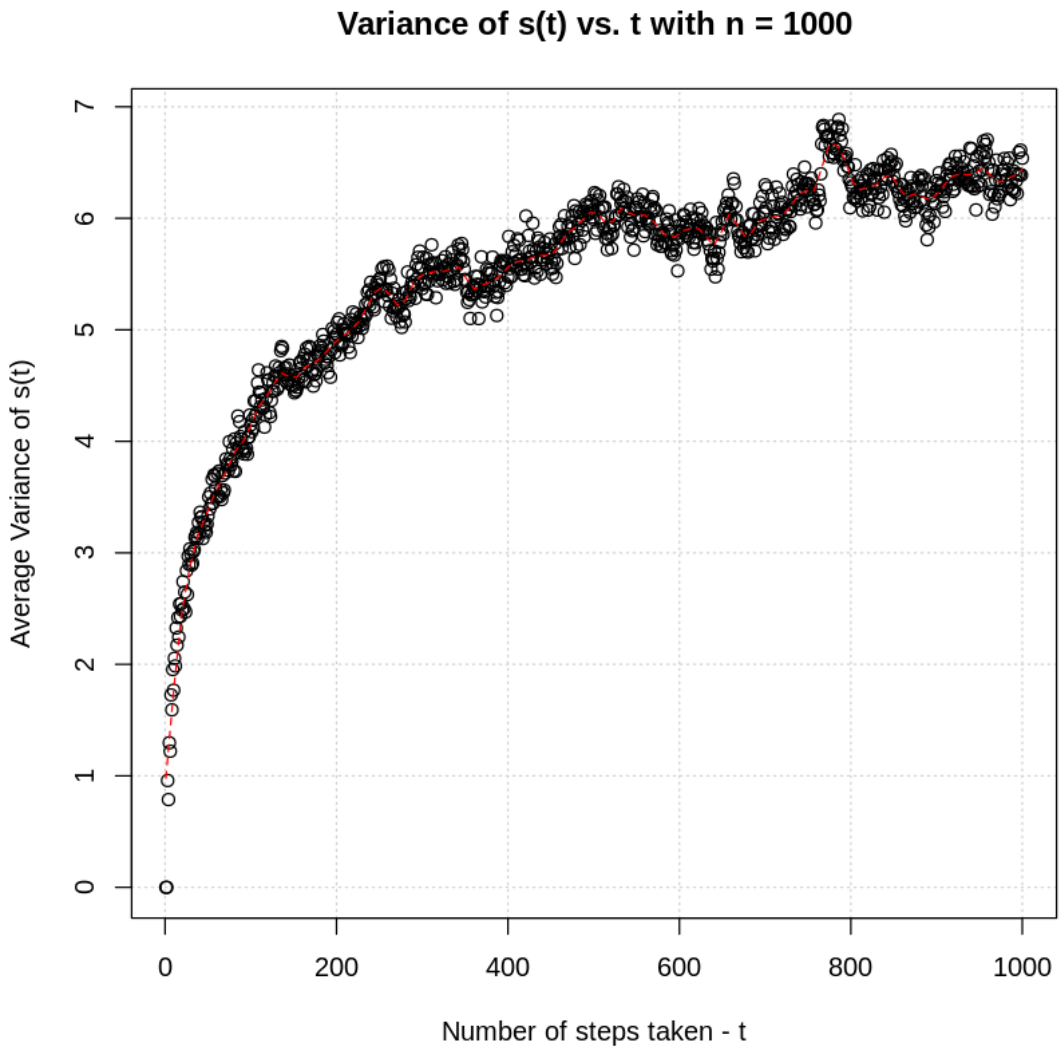
23 2(b)

[ ]:
```
s_t = matrix(data=0.0, nrow=1000, ncol=1000)
term_node_deg = matrix(data=0.0, nrow=1000, ncol=1)

for (i in 1:1000) {
    if (is_connected(g)) {
        gcc <- g
    }
    else {
        g.components <- clusters(g)
        ix <- which.max(g.components$csize)
        g.giant <- induced.subgraph(g, which(g.components$membership == ix))
        gcc <- g.giant
    }
    start = sample(V(gcc), 1)
    walked_nodes = random_walk(gcc, steps=1000, start)
    s_t[i,] = shortest.paths(gcc, walked_nodes, start)
    term_node_deg[i, 1] = degree(gcc, walked_nodes[length(walked_nodes)])
}
```

[ ]:
```
plot(seq(1, 1000, 1), colMeans(s_t), grid(), xlab="Number of steps taken - t",␣
 ↪ylab="Average Shortest Distance - <s(t)>", main="<s(t)> vs. t with n = 1000")
lines(lowess(seq(1, 1000, 1), colMeans(s_t), f=0.05), col="red", lwd=1, lty=2)
```

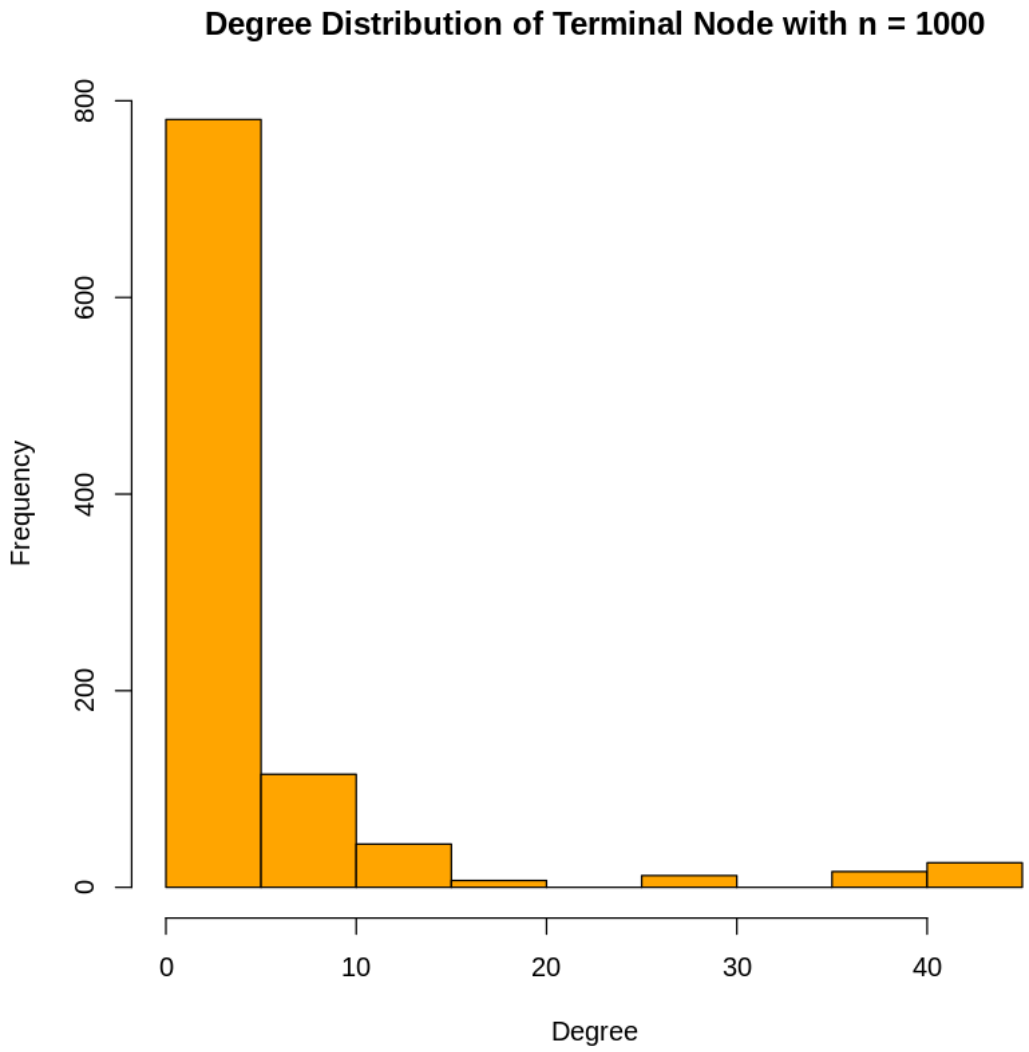


[ ]:
```
plot(seq(1, 1000, 1), colVars(s_t), grid(), xlab="Number of steps taken - t",␣
  ↪ylab="Average Variance of s(t)", main="Variance of s(t) vs. t with n = 1000")
lines(lowess(seq(1, 1000, 1), colVars(s_t), f=0.03), col="red", lwd=1, lty=2)
```

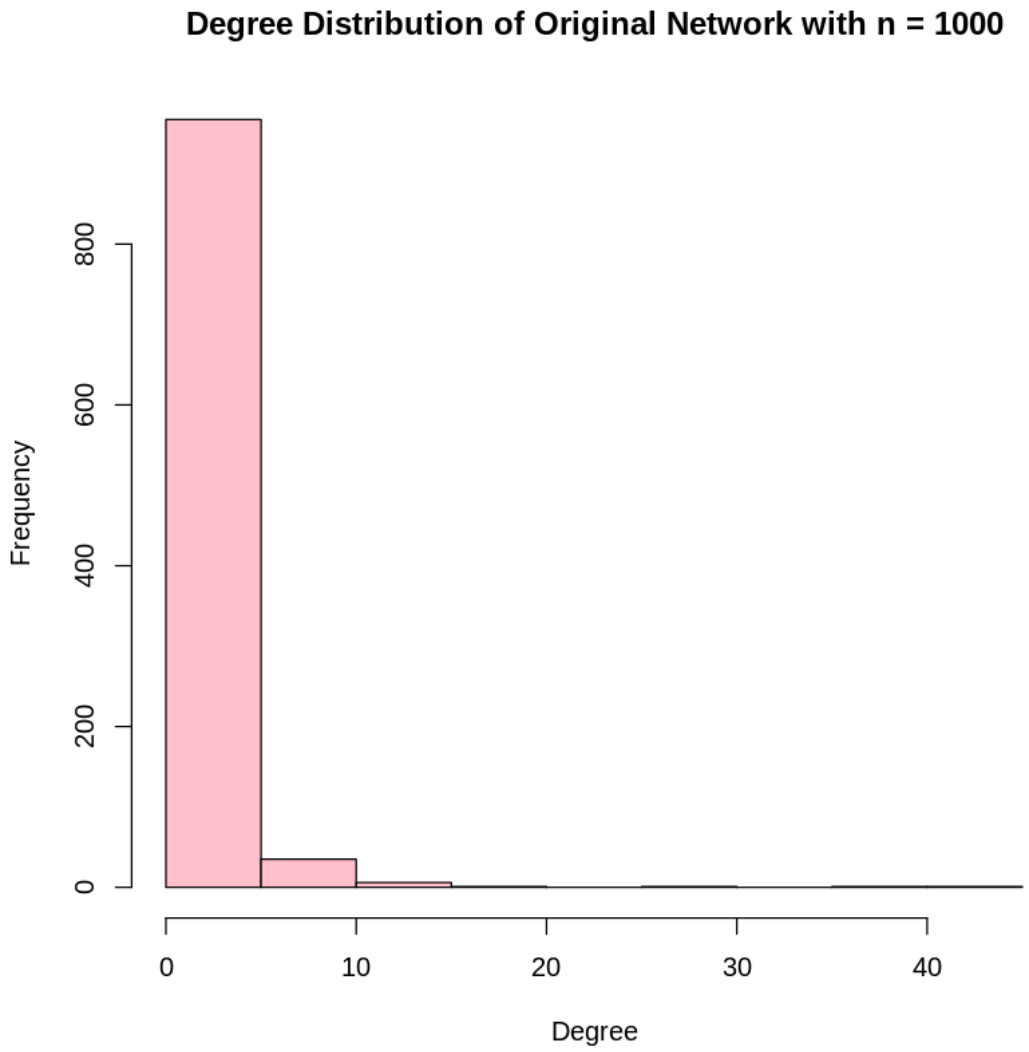


24 2(c)

[ ]:
```
hist(term_node_deg, col="orange", main="Degree Distribution of Terminal Node␣
↪with n = 1000", xlab="Degree", ylab="Frequency")
```



[ ]:
```
hist(degree(g), col="pink", main="Degree Distribution of Original Network with␣
↪n = 1000", xlab="Degree", ylab="Frequency")
```



[ ]:
```
print(diameter(g))
```

```
[1] 21
```

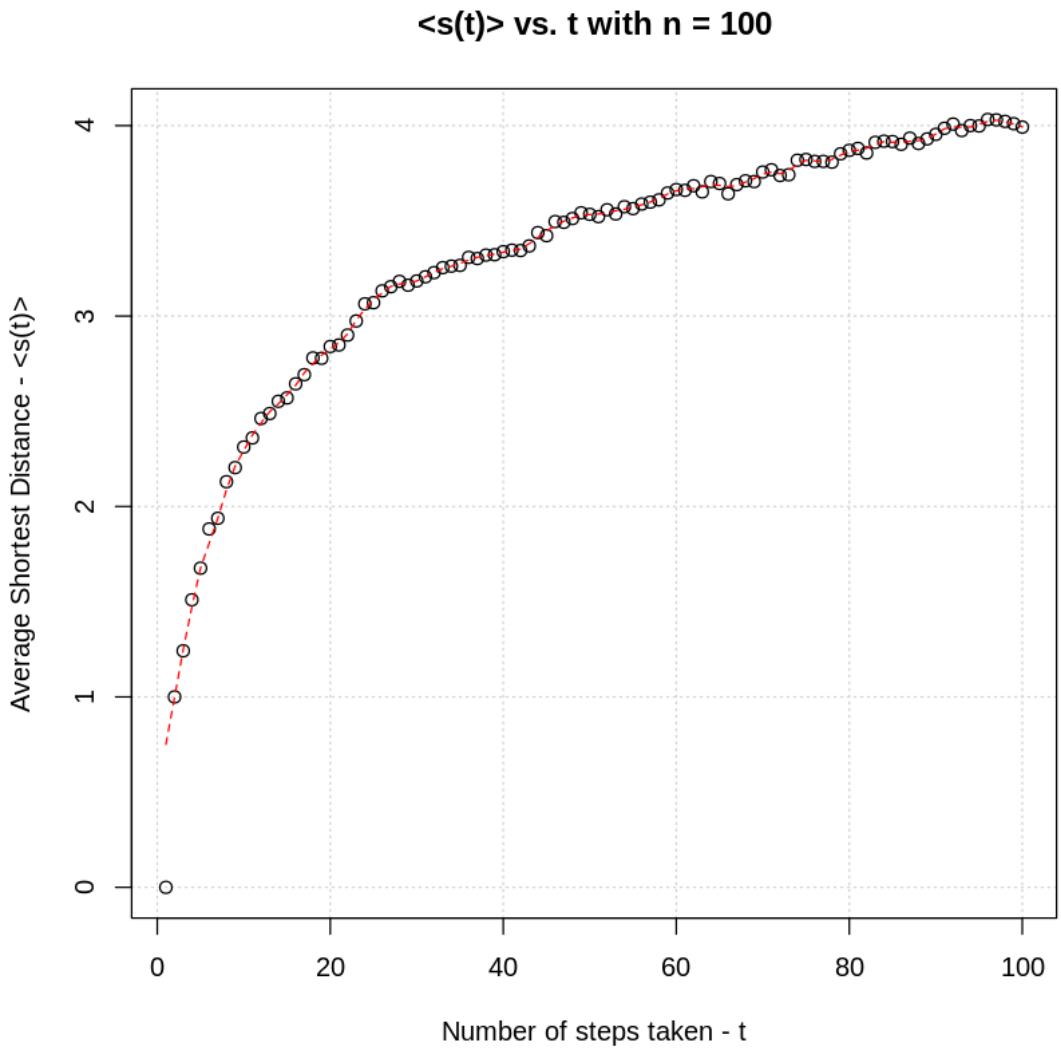
- Comparing these two graphs, we can see that the degree distribution of terminal node follows the degree distribution of entire network (fat-tailed power-law distribution): $P_k \propto \frac{1}{k^r} = \frac{1}{k^r \sum_{k=1}^{k_{max}} k^{-\gamma}}, k \in 1, 2, ..., k_{max}, \gamma > 0$
- Nodes with higher degree have a higher probability of getting selected as the terminal node.

25 2(d)

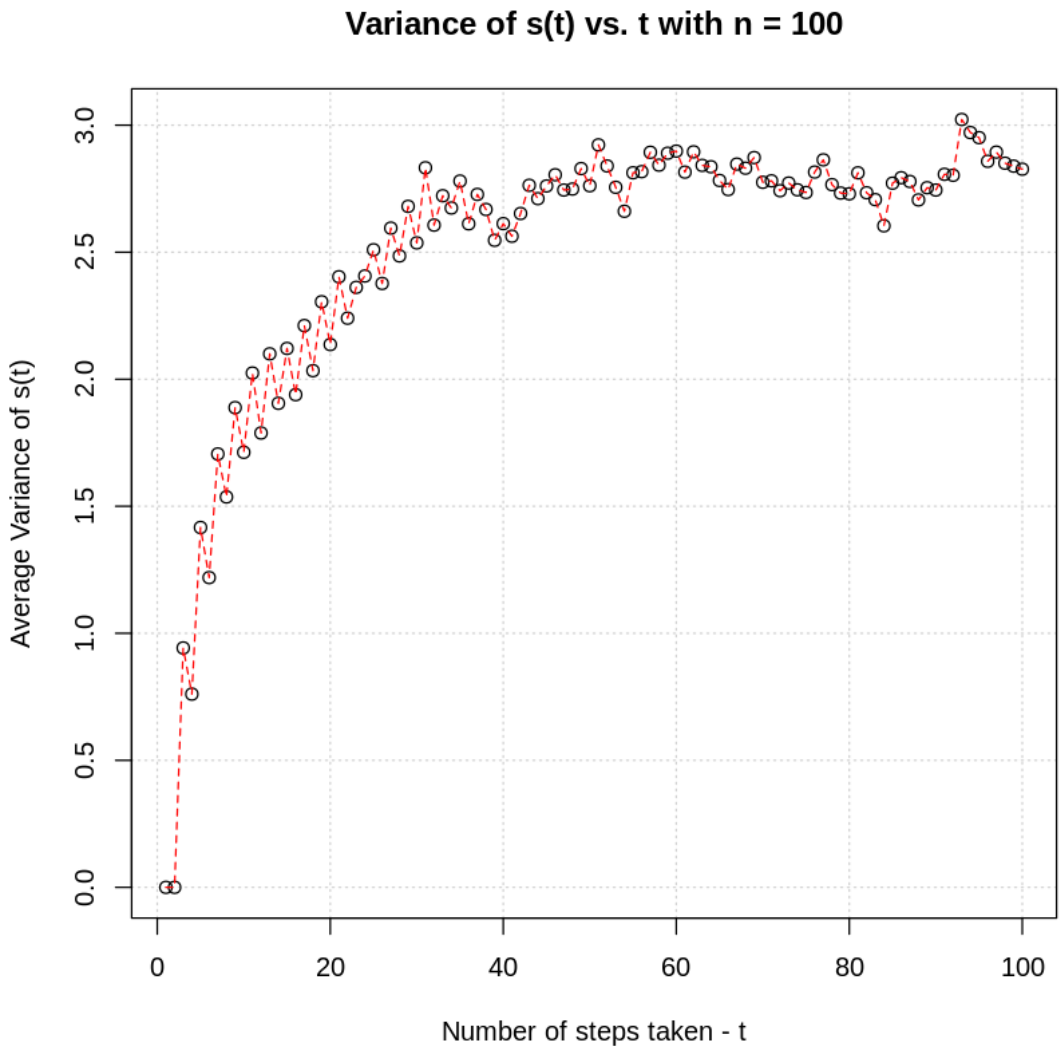
- 100 Nodes

[ ]:
```
g <- barabasi.game(100, m=1, directed=FALSE)
s_t = matrix(data=0.0, nrow=1000, ncol=100)
term_node_deg = matrix(data=0.0, nrow=1000, ncol=1)

for (i in 1:1000) {
    if (is_connected(g)) {
        gcc <- g
    }
    else {
        g.components <- clusters(g)
        ix <- which.max(g.components$csize)
        g.giant <- induced.subgraph(g, which(g.components$membership == ix))
        gcc <- g.giant
    }
    start = sample(V(gcc), 1)
    walked_nodes = random_walk(gcc, steps=100, start)
    s_t[i,] = shortest.paths(gcc, walked_nodes, start)
    term_node_deg[i, 1] = degree(gcc, walked_nodes[length(walked_nodes)])
}
```

[ ]:
```
plot(seq(1, 100, 1), colMeans(s_t), grid(), xlab="Number of steps taken - t",␣
 ↪ylab="Average Shortest Distance - <s(t)>", main="<s(t)> vs. t with n = 100")
lines(lowess(seq(1, 100, 1), colMeans(s_t), f=0.05), col="red", lwd=1, lty=2)
```



[ ]:
```
plot(seq(1, 100, 1), colVars(s_t), grid(), xlab="Number of steps taken - t",␣
  ↪ylab="Average Variance of s(t)", main="Variance of s(t) vs. t with n = 100")
lines(lowess(seq(1, 100, 1), colVars(s_t), f=0.03), col="red", lwd=1, lty=2)
```



[ ]:
```
print(diameter(g))
```

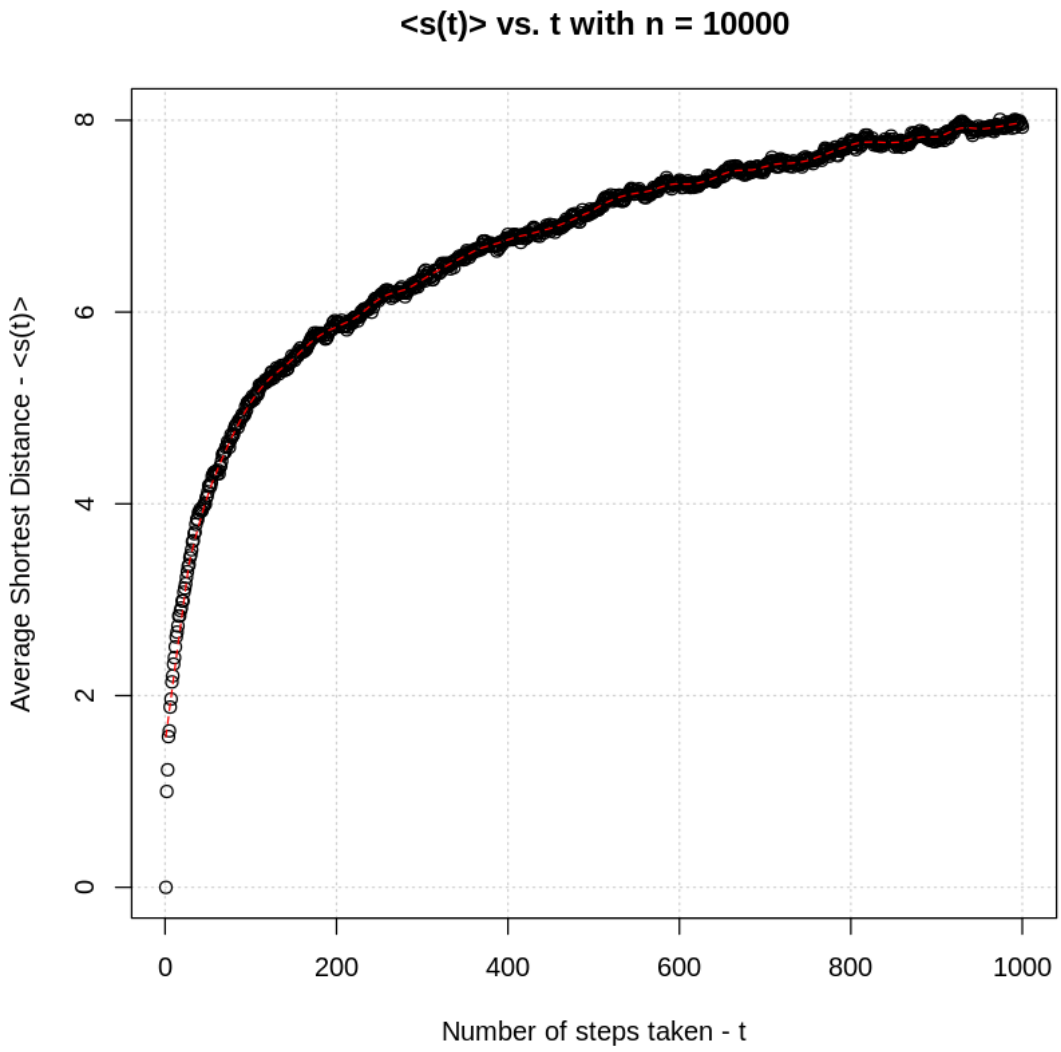
```
[1] 10
```

- 10000 Nodes

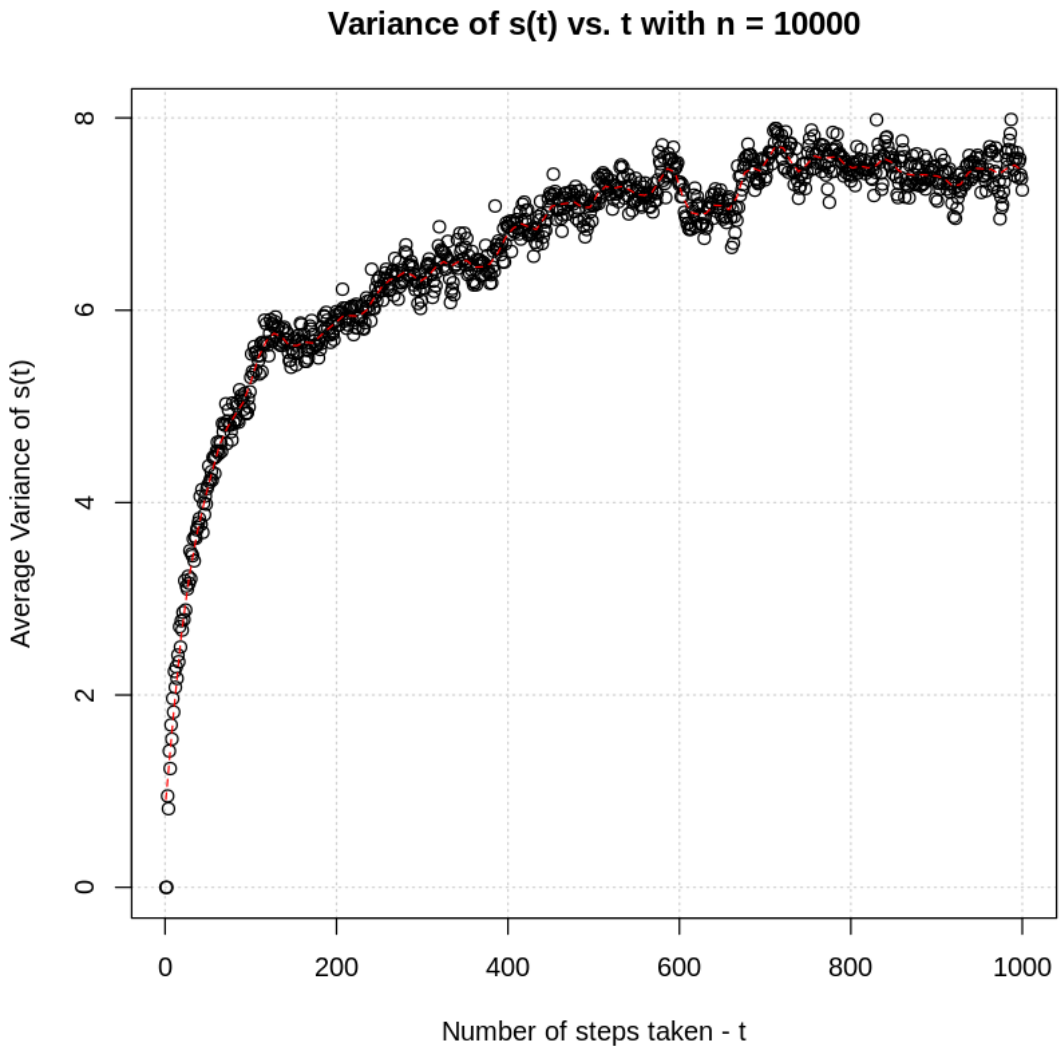
[ ]:
```
g <- barabasi.game(10000, m=1, directed=FALSE)
s_t = matrix(data=0.0, nrow=1000, ncol=1000)
term_node_deg = matrix(data=0.0, nrow=1000, ncol=1)

for (i in 1:1000) {
    if (is_connected(g)) {
        gcc <- g
    }
```

```
    else {
        g.components <- clusters(g)
        ix <- which.max(g.components$csize)
        g.giant <- induced.subgraph(g, which(g.components$membership == ix))
        gcc <- g.giant
    }
    start = sample(V(gcc), 1)
    walked_nodes = random_walk(gcc, steps=1000, start)
    s_t[i,] = shortest.paths(gcc, walked_nodes, start)
    term_node_deg[i, 1] = degree(gcc, walked_nodes[length(walked_nodes)])
}
```

[ ]:
```
plot(seq(1, 1000, 1), colMeans(s_t), grid(), xlab="Number of steps taken - t",␣
↪ylab="Average Shortest Distance - <s(t)>", main="<s(t)> vs. t with n =␣
↪10000")
lines(lowess(seq(1, 1000, 1), colMeans(s_t), f=0.05), col="red", lwd=1, lty=2)
```



[ ]:
```
plot(seq(1, 1000, 1), colVars(s_t), grid(), xlab="Number of steps taken - t",␣
  ↪ylab="Average Variance of s(t)", main="Variance of s(t) vs. t with n =␣
  ↪10000")
lines(lowess(seq(1, 1000, 1), colVars(s_t), f=0.03), col="red", lwd=1, lty=2)
```



[ ]:
```
print(diameter(g))
```

```
[1] 31
```

- Yes, the diameter of the network plays a role.
- The diameter of the network with 1000 nodes is 10, the diameter of the network with 1000 nodes is 21, and the diameter of the network with 10000 nodes is 31. The diameter of larger network (10000 nodes) is longer than the smaller network (100 nodes).
- The average shortest distance and the variance are larger for the larger networks. It is because the average path length scales logarithmically with the size of the network.
- The difference from the Erdos-Renyi network is that the diameter of the network increases with the increasing number of nodes. It is because larger networks are better capable of being partitioned into communities with strong intra-community connectedness and sparse inter-community connectedness.

- Since there are more communities, compared to smaller networks, random walkers of larger networks are more likely to end up at a high-degree node that is far away from the initial community. Therefore, with more number of nodes, the uncertainty in the shortest path between terminal nodes and initial nodes is also higher. It will result in a higher variance of shortest path values of larger networks and more number of steps to reach the steady-state.

26 3(a)

[2]:
```
create_transition_matrix = function (g){
    vs = V(g)
    n = vcount(g)
    adj = as_adjacency_matrix(g)
    adj[diag(rowSums(adj) == 0)] = 1
    z = matrix(rowSums(adj, , 1))
    transition_matrix = adj / repmat(z, 1, n)
    return(transition_matrix)
}
```

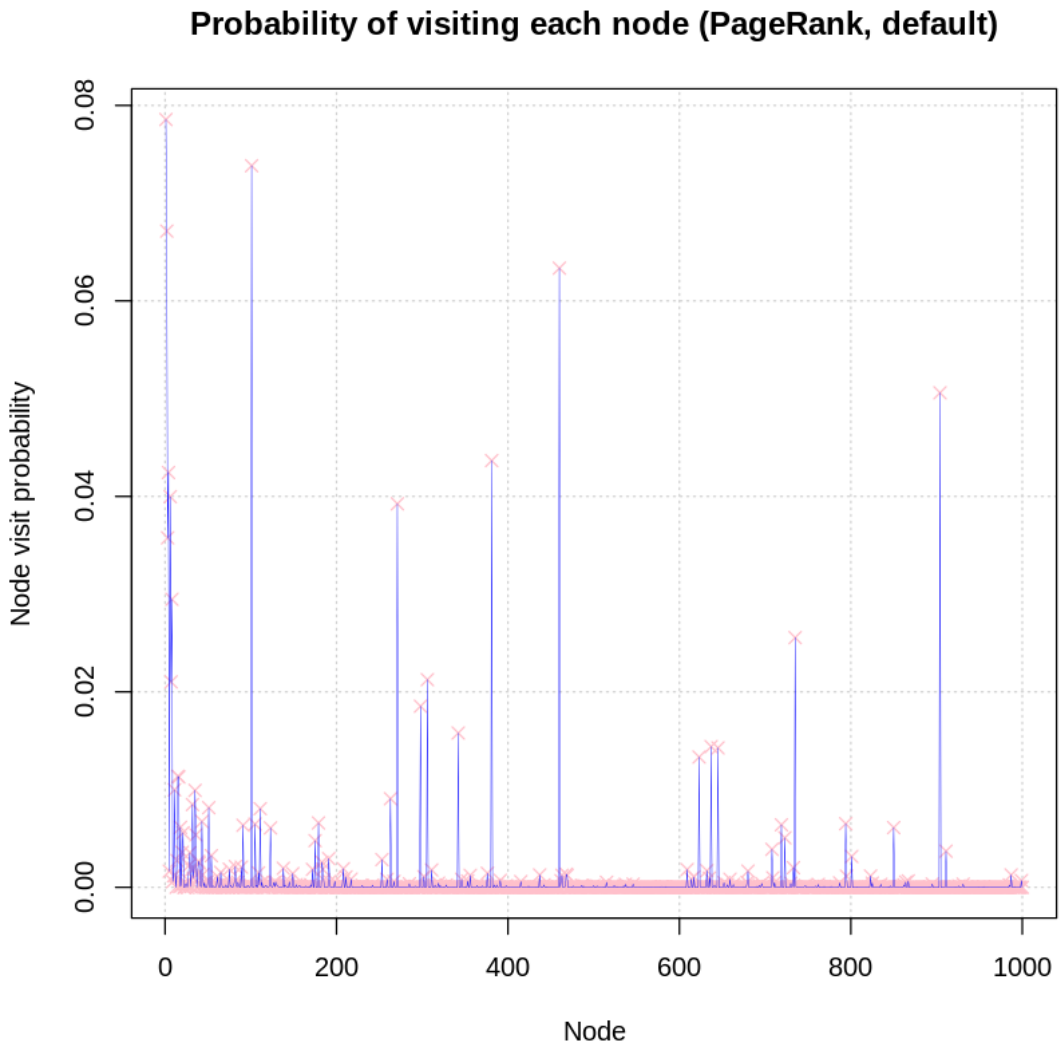
[3]:
```
random_walk_custom = function (g, num_steps, start_node,␣
 ↪transition_matrix=NULL, alpha=0.0, mode="equal"){
    if(is.null(transition_matrix)){
      transition_matrix = create_transition_matrix(g)
    }
    v = start_node
    walked_nodes = array(data=0.0, length(num_steps))

    if(mode=='equal'){
        visit_probs=array(1/vcount(g), vcount(g))
    }
    else if (mode=="page_rank"){
        pr = page_rank(g, directed=TRUE)$vector
        visit_probs= pr/sum(pr)
    }
    else if (mode=="median"){
        pr = page_rank(g, directed=TRUE)$vector
        df = data.frame("idx"=1:vcount(g), "val"=pr)
        df=df[order(df$val),]
        visit_probs=array(0, vcount(g))
        visit_probs[df$idx[(vcount(g1) %/% 2)]]=0.5
        visit_probs[df$idx[(vcount(g1) %/% 2)+1]]=0.5
    }

    for(i in 1:num_steps){
        if(runif(1)<alpha){
            v = sample(1:vcount(g), 1, prob = visit_probs)
        }
```

```
        else{
            PMF = transition_matrix[v, ]
            v = sample(1:vcount(g), 1, prob = PMF)
        }
        walked_nodes[i] = v
    }
    return(walked_nodes)
}
```

[6]:
```
g1 <- barabasi.game(1000, m = 4, directed = TRUE)
g2 <- barabasi.game(1000, m = 4, directed = TRUE)
gf = add_edges(g1, c(t(as_edgelist(permute(g2, sample(vcount(g2)))))))
```

[7]:
```
num_steps = 1000
visited_nodes=array(data=0.0, vcount(gf))
for(i in 1:100){
    start_node = sample(1:vcount(gf), 1)
    walked_nodes = random_walk_custom(g=gf, num_steps=num_steps,␣
 ↪transition_matrix=NULL, start_node=start_node, alpha=0.0, mode='equal')
    for (j in 1:length(walked_nodes)) {
        visited_nodes[walked_nodes[j]] = visited_nodes[walked_nodes[j]] + 1
    }
}
node_prob = visited_nodes / (num_steps * 100)
```

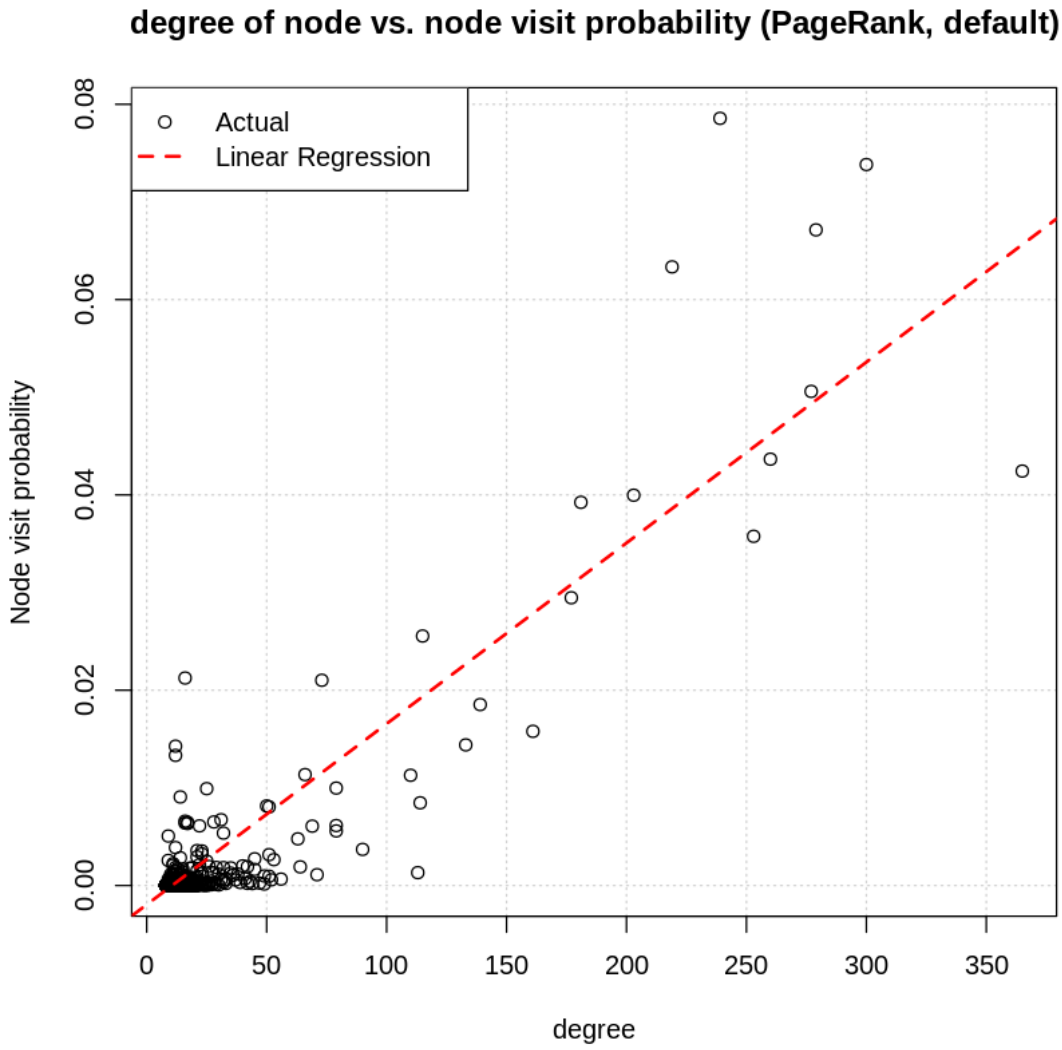
[8]:
```
plot(seq(1,1000,1),node_prob,pch=4, xlab='Node', ylab='Node visit␣
 ↪probability',main='Probability of visiting each node (PageRank,␣
 ↪default)',col='pink',grid())
lines(node_prob,lwd=0.3,col="blue")
```



[9]:
```
plot(degree(gf), node_prob , xlab='degree', ylab='Node visit␣
 ↪probability',main="degree of node vs. node visit probability (PageRank,␣
 ↪default)",grid())
abline(lm(node_prob ~ degree(gf)),col="red",lwd=2,lty=2)
legend('topleft', legend = c("Actual","Linear Regression"),
       lty = c(NA, 2), lwd = c(1,2), pch=c(1,NA),
       col = c('black','red'))
print(sprintf("Pearson correlation coefficient: %f",cor(degree(gf), node_prob)))
print("Slope and intercept:")
print(lm(node_prob ~ degree(gf)))
```

```
[1] "Pearson correlation coefficient: 0.896892"
[1] "Slope and intercept:"
```

```
Call:
lm(formula = node_prob ~ degree(gf))

Coefficients:
(Intercept)   degree(gf)
 -0.0019553    0.0001852
```

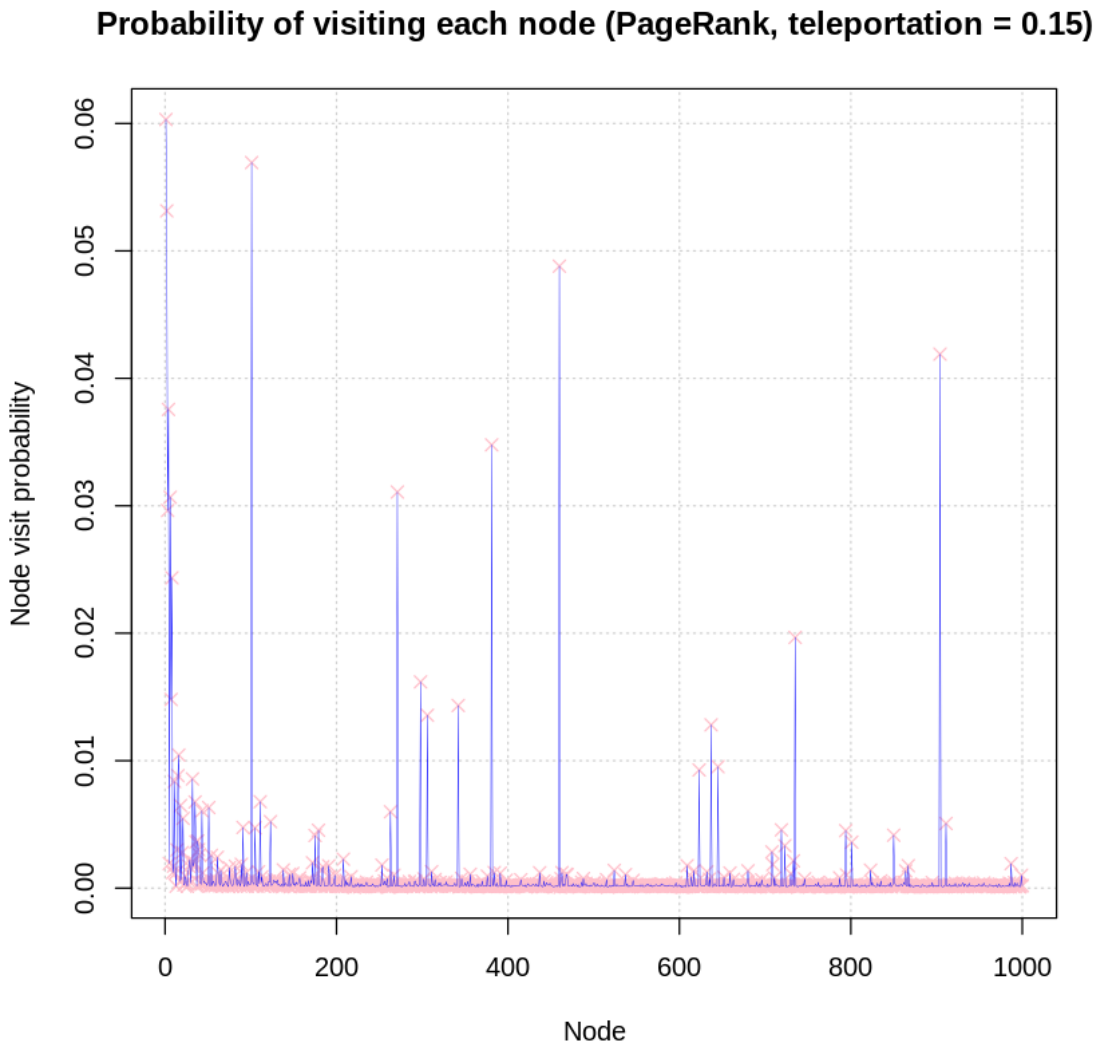


From the figure above, we can see that degree of nodes and visit probability are highly correlated with a correlation coefficient of 0.896892. This makes sense since, in preferential attachment networks, nodes with higher degrees have a higher probability to be selected during the random walk process.

27 3(b)

[11]:
```
num_steps = 1000
visited_nodes=array(data=0.0, vcount(gf))
for(i in 1:100){
    start_node = sample(1:vcount(gf), 1)
    walked_nodes = random_walk_custom(g=gf, num_steps=num_steps,␣
 ↪transition_matrix=NULL, start_node=start_node, alpha=0.15, mode='equal')
    for (j in 1:length(walked_nodes)) {
            visited_nodes[walked_nodes[j]] = visited_nodes[walked_nodes[j]] + 1
    }
}
node_prob = visited_nodes / (num_steps * 100)
```

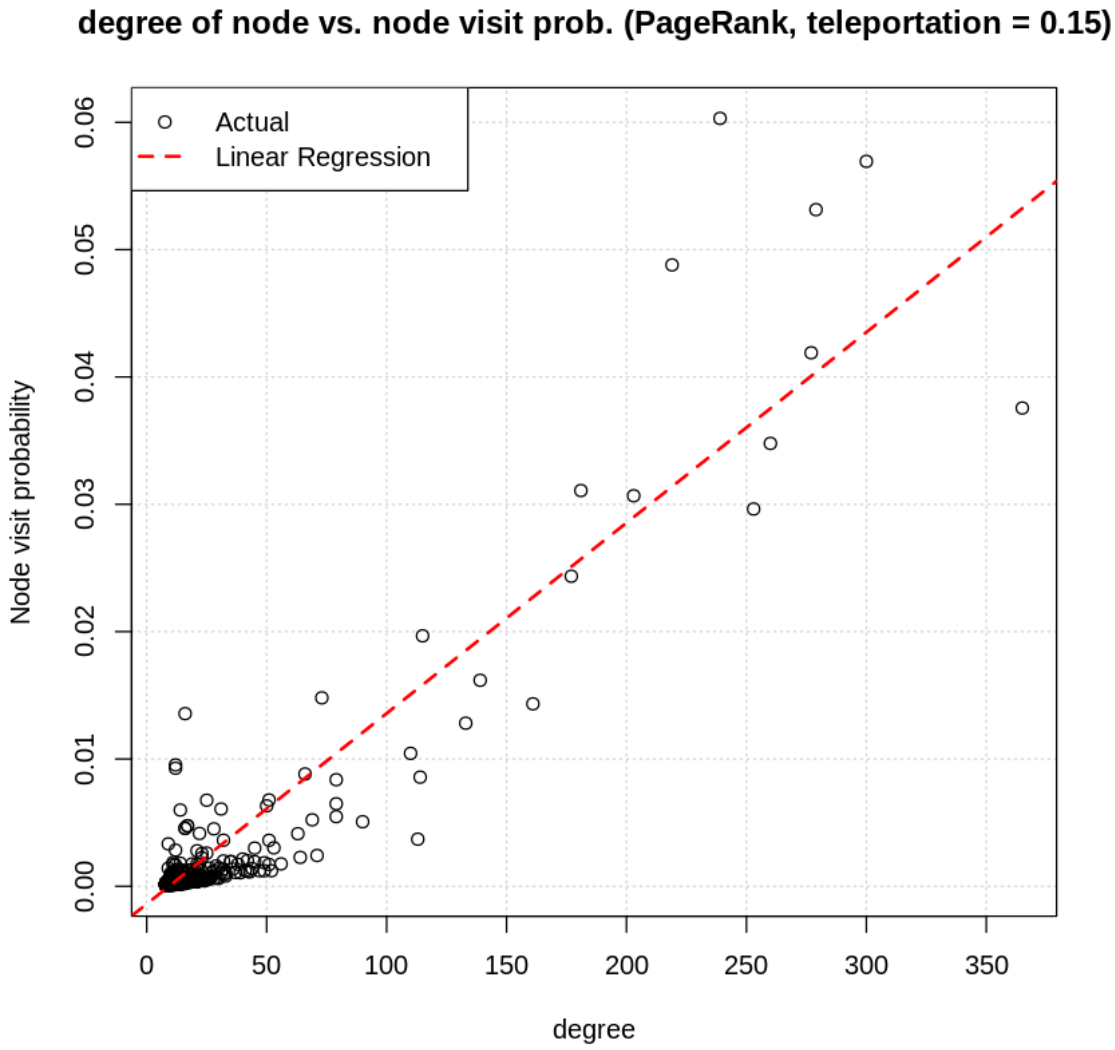
[12]:
```
plot(seq(1,1000,1),node_prob,pch=4, xlab='Node', ylab='Node visit␣
 ↪probability',main='Probability of visiting each node (PageRank,␣
 ↪teleportation = 0.15)',col='pink',grid())
lines(node_prob,lwd=0.3,col="blue")
```



[13]:
```
plot(degree(gf), node_prob , xlab='degree', ylab='Node visit␣
 ↪probability',main="degree of node vs. node visit prob. (PageRank,␣
 ↪teleportation = 0.15)",grid())
abline(lm(node_prob ~ degree(gf)),col="red",lwd=2,lty=2)
legend('topleft', legend = c("Actual","Linear Regression"),
       lty = c(NA, 2), lwd = c(1,2), pch=c(1,NA),
       col = c('black','red'))
print(sprintf("Pearson correlation coefficient: %f",cor(degree(gf), node_prob)))
print("Slope and intercept:")
print(lm(node_prob ~ degree(gf)))
```

```
[1] "Pearson correlation coefficient: 0.922639"
[1] "Slope and intercept:"
```

```
Call:
lm(formula = node_prob ~ degree(gf))

Coefficients:
(Intercept)   degree(gf)
 -0.0013888    0.0001497
```

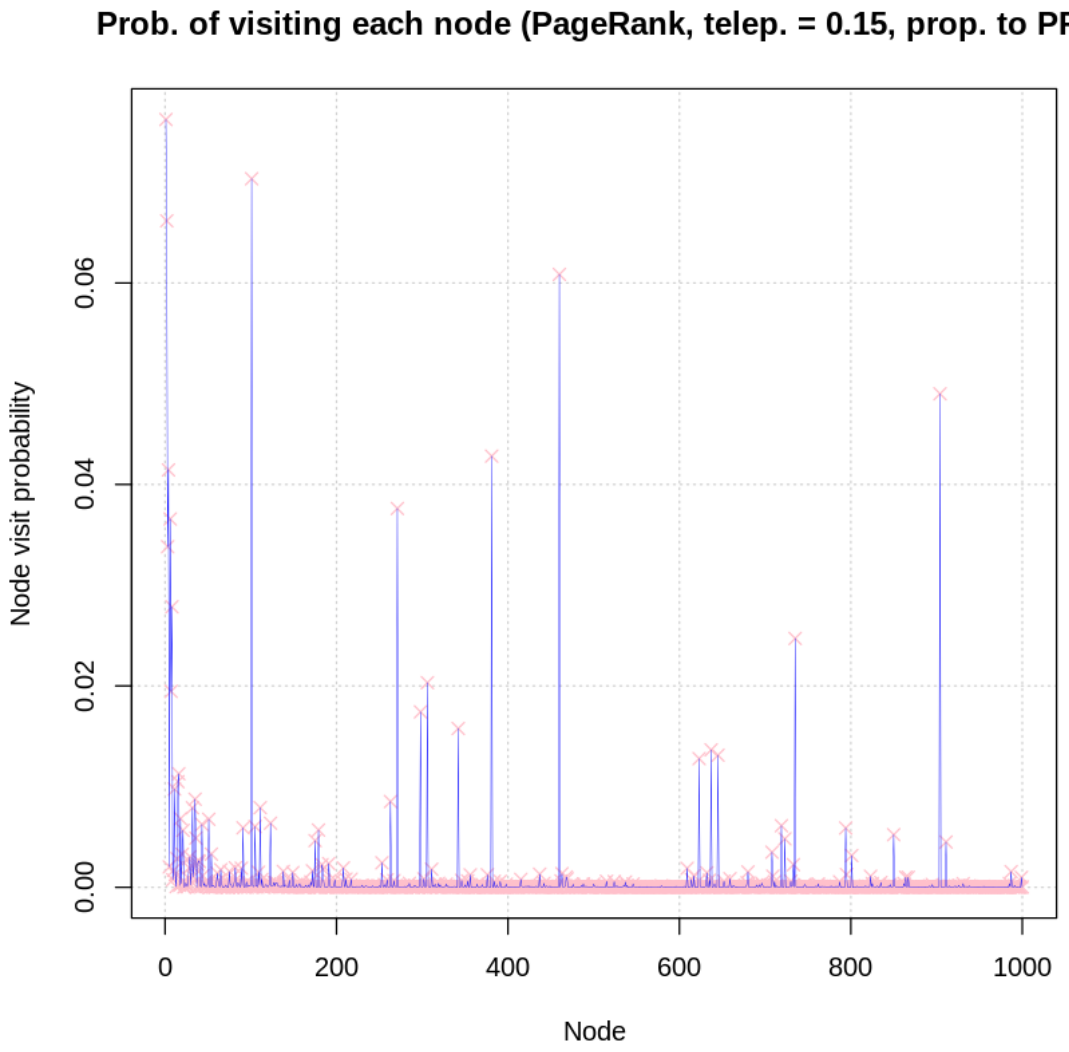


Similarly, we can see that degree of nodes and visit probability are highly correlated as 3(a). Also, we can observe that the slope is lower than no teleportation($0.0001497 < 0.0001852$). That is to say, the property of teleportation reduces the effect made by the degree of nodes because with a probability of 0.15 the walker would randomly choose some node.

28 4(a)

[14]:
```
num_steps = 1000
visited_nodes=array(data=0.0, vcount(gf))
for(i in 1:100){
    start_node = sample(1:vcount(gf), 1)
    walked_nodes = random_walk_custom(g=gf, num_steps=num_steps,␣
 ↪transition_matrix=NULL, start_node=start_node, alpha=0.15, mode='page_rank')
    for (j in 1:length(walked_nodes)) {
            visited_nodes[walked_nodes[j]] = visited_nodes[walked_nodes[j]] + 1
    }
}
node_prob = visited_nodes / (num_steps * 100)
```

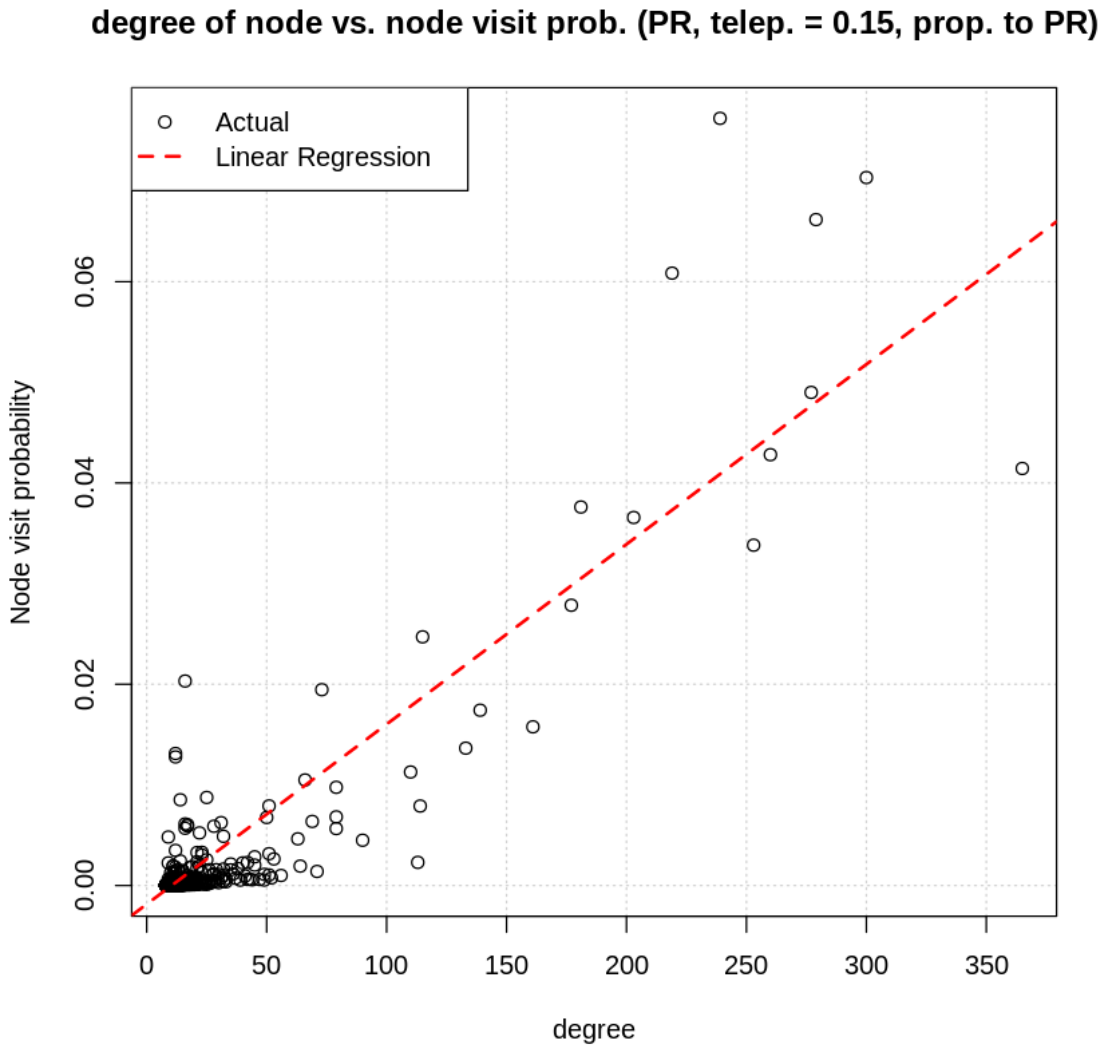
[15]:
```
plot(seq(1,1000,1),node_prob,pch=4, xlab='Node', ylab='Node visit␣
 ↪probability',main='Prob. of visiting each node (PageRank, telep. = 0.15,␣
 ↪prop. to PR)',col='pink',grid())
lines(node_prob,lwd=0.3,col="blue")
```



[16]:
```
plot(degree(gf), node_prob , xlab='degree', ylab='Node visit␣
 ↪probability',main="degree of node vs. node visit prob. (PR, telep. = 0.15,␣
 ↪prop. to PR)",grid())
abline(lm(node_prob ~ degree(gf)),col="red",lwd=2,lty=2)
legend('topleft', legend = c("Actual","Linear Regression"),
       lty = c(NA, 2), lwd = c(1,2), pch=c(1,NA),
       col = c('black','red'))
print(sprintf("Pearson correlation coefficient: %f",cor(degree(gf), node_prob)))
print("Slope and intercept:")
print(lm(node_prob ~ degree(gf)))
```

```
[1] "Pearson correlation coefficient: 0.900486"
[1] "Slope and intercept:"
```

```
Call:
lm(formula = node_prob ~ degree(gf))

Coefficients:
(Intercept)   degree(gf)
 -0.0018540    0.0001788
```

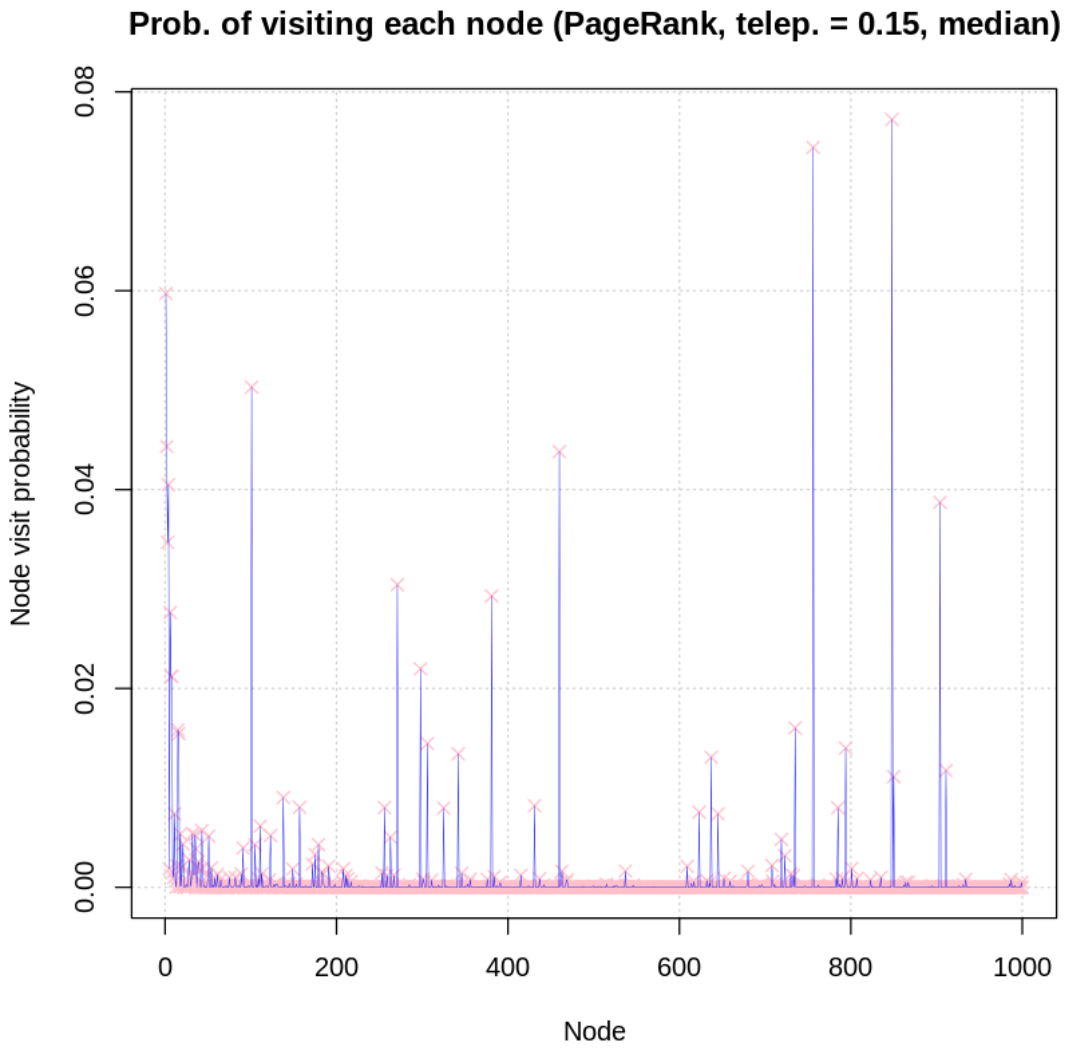


We know teleportation would reduce the slope and personalized PageRank would increase the slope since it follows the PR value to choose the next node for teleportation. Hence, we can see that the slope is larger than 3(b) ($0.0001788 > 0.0001497$). However, the slope is still less than 3(a) which means the effect caused by teleportation is more influential.

29 4(b)

[18]:
```
num_steps = 1000
visited_nodes=array(data=0.0, vcount(gf))
for(i in 1:100){
    start_node = sample(1:vcount(gf), 1)
    walked_nodes = random_walk_custom(g=gf, num_steps=num_steps,␣
 ↪transition_matrix=NULL, start_node=start_node, alpha=0.15, mode='median')
    for (j in 1:length(walked_nodes)) {
            visited_nodes[walked_nodes[j]] = visited_nodes[walked_nodes[j]] + 1
    }
}
node_prob = visited_nodes / (num_steps * 100)
```

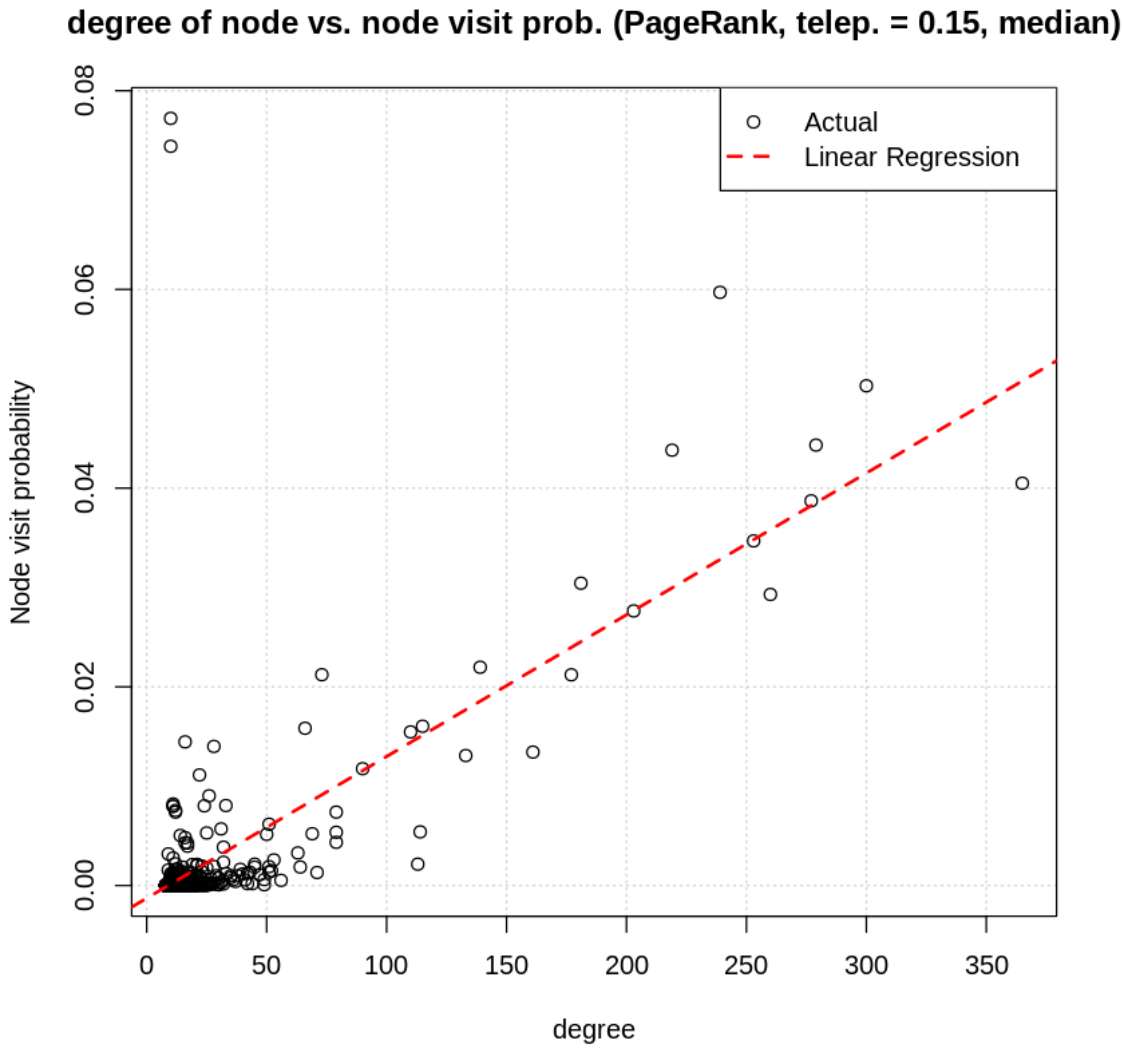
[19]:
```
plot(seq(1,1000,1),node_prob,pch=4, xlab='Node', ylab='Node visit␣
 ↪probability',main='Prob. of visiting each node (PageRank, telep. = 0.15,␣
 ↪median)',col='pink',grid())
lines(node_prob,lwd=0.3,col="blue")
```



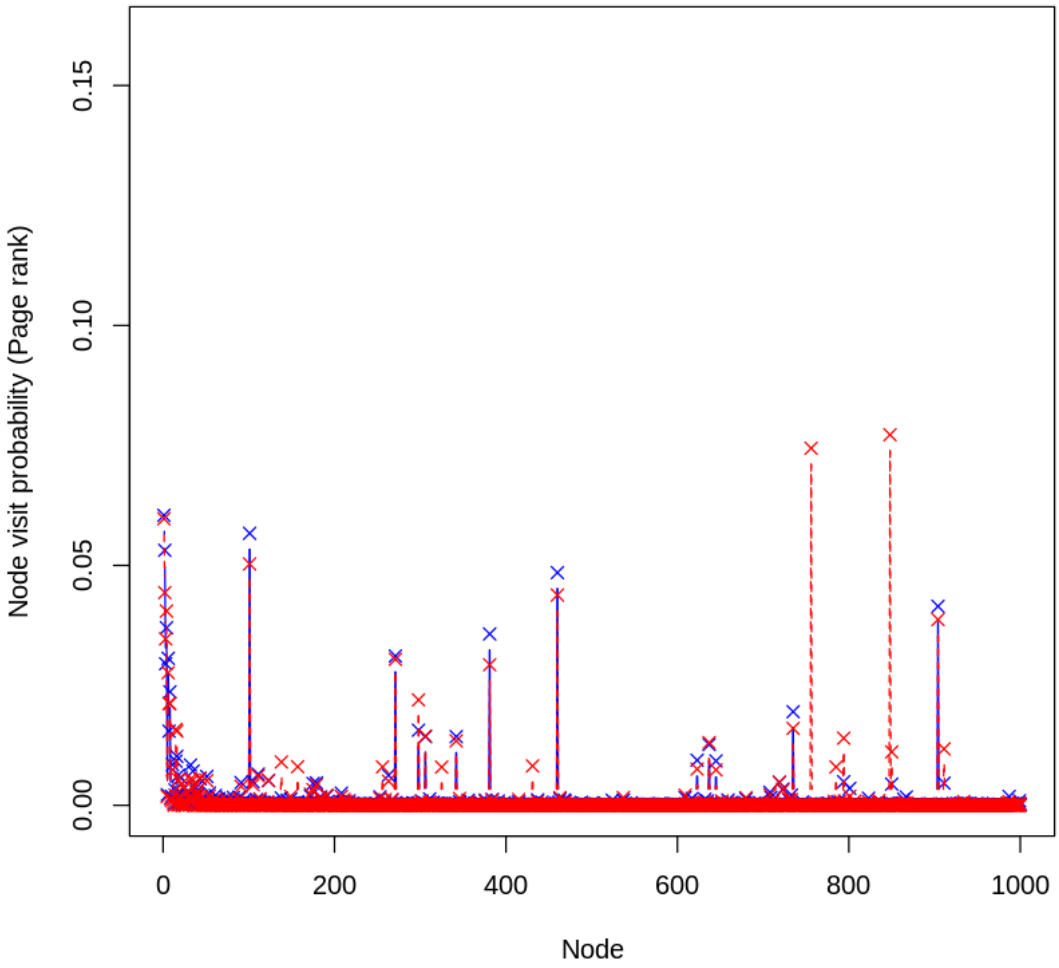
[20]:
```
plot(degree(gf), node_prob , xlab='degree', ylab='Node visit␣
 ↪probability',main="degree of node vs. node visit prob. (PageRank, telep. = 0.
 ↪15, median)",grid())
abline(lm(node_prob ~ degree(gf)),col="red",lwd=2,lty=2)
legend('topright', legend = c("Actual","Linear Regression"),
       lty = c(NA, 2), lwd = c(1,2), pch=c(1,NA),
       col = c('black','red'))
print(sprintf("Pearson correlation coefficient: %f",cor(degree(gf), node_prob)))
print("Slope and intercept:")
print(lm(node_prob ~ degree(gf)))
```

```
[1] "Pearson correlation coefficient: 0.725743"
[1] "Slope and intercept:"
```

```
Call:
lm(formula = node_prob ~ degree(gf))

Coefficients:
(Intercept)   degree(gf)
 -0.0012765    0.0001426
```



[21]:
```
pageranks = page_rank(gf)$vector
plot(seq(1,1000,1),pageranks,col="blue",type='b',pch=4,ylim=c(0,0.16),␣
  ↪xlab='Node', ylab='Node visit probability (Page rank)')
points(node_prob, col="red",type='b',lty=2,pch=4)
```



From the figure above, we can see that the PR value converges to fewer nodes. Since we only have two choices with teleportation which means we would visit these two nodes more often and the nodes near these two nodes.

30 4(c)

Let T is the set of trusted nodes, then

$$\pi(i) = (1-\alpha)\sum_{j=1}^{|V|}\frac{1}{k_{out}(j)}A_{ji}\pi(j) + \frac{\alpha}{|T|}, i \in T$$

$$\pi(i) = (1-\alpha)\sum_{j=1}^{|V|}\frac{1}{k_{out}(j)}A_{ji}\pi(j), i \notin T$$