Part1_Q2

April 15, 2020

[2]:
```
# # Install Necessary Packages
install.packages("igraph")
install.packages("Matrix")
install.packages("pracma")

# Load Packages
library('igraph')
library('Matrix')
library('pracma')
```

```
Updating HTML index of packages in '.Library'
Making 'packages.html' … done
Updating HTML index of packages in '.Library'
Making 'packages.html' … done
Updating HTML index of packages in '.Library'
Making 'packages.html' … done

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
```

[4]:
```
### 2a) Undirected Network with Preferential attachment.

#Generate an undirected network with preferential attachment.
```

```
n = 1000
m = 1
g <- barabasi.game(n,m, directed=F);
print(sprintf("Is the graph always connected : %s",is.connected(g)))
# g <- set.graph.attribute(g, "layout", layout.fruchterman.reingold(g))
png(sprintf("plots/part1/question2/q2a.png"))
plot(g, vertex.label="", vertex.size=2, main=sprintf("Undirected Network with␣
 ↪preferential attachment (n=%d, m=%d)",n,m))
dev.off()
```

```
[1] "Is the graph always connected : TRUE"
```

png: 2

[7]:
```
# 2b) Find Community Structure and Modularity.

# Find the community structure
g_comm = cluster_fast_greedy(g)
#print(paste("Community is: ", g_comm))

# Find the size of community
community_size = sizes(g_comm)
#cat(paste("Community Size is:", community_size))

# Find modularity
g_modularity = modularity(g_comm)
cat(paste("\nModularity is ", g_modularity))

# Plot graph with communities
png(sprintf("plots/part1/question2/q2b.png"))
plot(g_comm, g, main=sprintf("Community Structure (n=%d, m=%d)",n,m),vertex.
 ↪size=2, vertex.label=NA)
dev.off()
```

```
Modularity is  0.934286137989844
```

png: 2

[9]:
```
# 2c) Undirected Network with Preferential attachment.

#Generate an undirected network with preferential attachment.
n = 10000
m = 1
g <- barabasi.game(n,m, directed=F);
print(sprintf("Is the graph always connected : %s",is.connected(g)))
# g <- set.graph.attribute(g, "layout", layout.fruchterman.reingold(g))
png(sprintf("plots/part1/question2/q2c_network.png"))
```

```
plot(g, vertex.label="", vertex.size=2, main=sprintf("Undirected Network with␣
 ↪preferential attachment (n=%d, m=%d)",n,m))
dev.off()

# Find the community structure
g_comm = cluster_fast_greedy(g)
# print(paste("Community is: ", g_comm))

# Find the size of community
community_size = sizes(g_comm)
# cat(paste("Community Size is:", community_size))

# Find modularity
g_modularity = modularity(g_comm)
cat(paste("\nModularity is ", g_modularity))

# Plot graph with communities
png(sprintf("plots/part1/question2/q2c_community_structure.png"))
plot(g_comm, g, main=sprintf("Community Structure (n=%d, m=%d)",n,m),vertex.
 ↪size=2, vertex.label=NA)
dev.off()
```

```
[1] "Is the graph always connected : TRUE"
```

png: 2

```
Modularity is  0.978358131842797
```

png: 2

[10]:
```
#2d) Plot degree distribution in log-log scale.

n_array = c(1000,10000)
m=1
for (n in n_array) {
    g = barabasi.game(n=n,m=m, directed=F)
    deg_distribution = degree.distribution(g)
    # take log
    idx = which(deg_distribution != 0, arr.ind=TRUE) #remove 0s
    x = log(seq(1:length(deg_distribution)))[idx]
    y = log(deg_distribution)[idx]

    cat(paste("For n = ", n))

    # Solve linear Equation:
    relation = lm(y ~ x)
    print(relation)
    png(sprintf("plots/part1/question2/q2d_logplot_n_%d.png",n))
```

```
    plot(x,y,abline(relation,col="red"),main=sprintf("Degree distribution of␣
 ↪the network for n=%d (log-log␣
 ↪plot)",n),xlab="log(Degree)",ylab="log(Probability)")
    dev.off()
    png(sprintf("plots/part1/question2/q2d_hist_n_%d.png",n))
    hist(degree(g),col=rgb(0.1,0.5,1,.6),main=sprintf("Degree distribution of␣
 ↪the network for n=%d (Histogram)",n),xlab="Degree",ylab="Frequency" )
    dev.off()
}
```

```
For n =  1000
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     0.6869      -2.4369

For n =  10000
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     0.7856      -2.7255
```

[13]:
```
#2e) log degree distribution of random jth neighbour

n_array = c(1000,10000)
m = 1
iteration_times = 10000
for (n in n_array) {

    g = barabasi.game(n=n, m=m, directed=FALSE)
    degree_neighbors = c()
    for (i in seq(1:iteration_times)) # Reapeat 10000 times
    {
        node_i = sample(n, 1) # Get a random number
        neighbors_node_i = neighbors(g, node_i)
        if (length(neighbors_node_i) == 1)
        {
            node_j = neighbors_node_i
        }
        else
        {
            node_j = sample(neighbors_node_i, 1)
```

```
        }
        degree_neighbors = c(degree_neighbors, degree(g, node_j))
    }
    cat(paste("\n results for n=",n))
    png(sprintf("plots/part1/question2/q2e_hist_n_%d.png",n))
    h = hist(degree_neighbors, breaks=seq(0, max(degree_neighbors)),␣
↪freq=FALSE, main =sprintf("Histogram of degree distribution (n=%d)",n),␣
↪xlab="Degree")
    #h = hist(degree_neighbors, main =sprintf("Histogram of degree distribution␣
↪for jth node with n=%d",n), xlab="Degree", ylab="Frequency")
    dev.off()
    plot_x = tail(h$breaks, length(h$breaks) - 1) # Remove 0
    plot_y = h$density
    idx = which(plot_y != 0, arr.ind=TRUE)
    log_degree = log(seq(1:length(plot_y)))[idx]
    log_distribution = log(plot_y)[idx]
    relation = lm(log_distribution ~ log_degree)
    print(relation)

    png(sprintf("plots/part1/question2/q2e_logplot_n_%d.png",n))
    plot(log_degree, log_distribution,abline(relation,col="red"),␣
↪main=sprintf("Degree distribution of the jth node for n=%d m=%d (log-log␣
↪plot)",n,m), xlab="log(Degree)", ylab="log(Density)")
    dev.off()
}
```

```
 results for n= 1000
Call:
lm(formula = log_distribution ~ log_degree)

Coefficients:
(Intercept)   log_degree
     -1.022       -1.229


 results for n= 10000
Call:
lm(formula = log_distribution ~ log_degree)

Coefficients:
(Intercept)   log_degree
    -0.7553      -1.4900
```

[14]:
```
#2f) Age of nodes
```

```
n = 1000
m = 1
iteration_times =1000
degrees.sum = vector(mode = "numeric", length = iteration_times)
for (i in 1:iteration_times){
    g = barabasi.game(n=n,m = m, directed=F)
    degrees.sum = degrees.sum + degree(g)
}

png(sprintf("plots/part1/question2/q2f.png"))
plot(seq(1:iteration_times),rev(degrees.sum/iteration_times) ,main=sprintf("Age␣
 ↪of Nodes vs Expected Degree (n=%d m=%d)",n,m),xlab="Age of␣
 ↪Nodes",ylab="Expected Degree",type="l")
dev.off()
```

png: 2

[15]:
```
#2g) Repeat 2a-2f for m=2, m=5

m_array = c(2,5)
n_array = c(1000,10000)

for (m in m_array) {

    print("2a, 2b, 2c")
    for (n in n_array){
        cat(paste("\nn= ", n))
        cat(paste("\nm= ", m))
        g <- barabasi.game(n =n,m = m, directed=F);
        cat("\n")
        print(sprintf("Is the graph always connected : %s",is.connected(g)))
        png(sprintf("plots/part1/question2/q2g_partabc_network_n%d_m%d.
 ↪png",n,m))
        plot(g, vertex.label="", vertex.size=2, main=sprintf("Undirected␣
 ↪Network with preferential attachment(n=%d m=%d)",n,m))
        dev.off()
        g_comm = cluster_fast_greedy(g)
        community_size = sizes(g_comm)
        g_modularity = modularity(g_comm)
        cat(paste("\nModularity is ", g_modularity))
        png(sprintf("plots/part1/question2/q2g_partabc_commstruct_n%d_m%d.
 ↪png",n,m))
        plot(g_comm, g, main=sprintf("Community Structure (n=%d␣
 ↪m=%d)",n,m),vertex.size=2, vertex.label=NA)
        dev.off()
    }
```

```
   print("2d")
   for (n in n_array) {
       cat(paste("\nn= ", n))
       cat(paste("\nm= ", m))
       g = barabasi.game(n=n,m=m, directed=F)
       deg_distribution = degree.distribution(g)
       idx = which(deg_distribution != 0, arr.ind=TRUE)
       x = log(seq(1:length(deg_distribution)))[idx]
       y = log(deg_distribution)[idx]
       relation = lm(y ~ x)
       cat("\n")
       print(relation)
       png(sprintf("plots/part1/question2/q2g_partd_logplot_n%d_m%d.png",n,m))
       plot(x,y,abline(relation,col="red"),main=sprintf("Degree distribution␣
↪of the network for n=%d, m=%d (log-log␣
↪plot)",n,m),xlab="log(Degree)",ylab="log(Probability)")
       dev.off()
       png(sprintf("plots/part1/question2/q2g_partd_hist_n%d_m%d.png",n,m))
       hist(degree(g),col=rgb(0.1,0.5,1,.6),main=sprintf("Degree distribution␣
↪of the network for n=%d, m=%d␣
↪(Histogram)",n,m),xlab="Degree",ylab="Frequency" )
       dev.off()
   }

   print("2e")
   iteration_times = 10000
   for (n in n_array) {
       cat(paste("\nn= ", n))
       cat(paste("\nm= ", m))
       g = barabasi.game(n=n, m=m, directed=FALSE)
       degree_neighbors = c()
       for (i in seq(1:iteration_times)){
           node_i = sample(n, 1)
           neighbors_node_i = neighbors(g, node_i)
           if (length(neighbors_node_i) == 1)
           {
               node_j = neighbors_node_i
           }
           else
           {
               node_j = sample(neighbors_node_i, 1)
           }
           degree_neighbors = c(degree_neighbors, degree(g, node_j))
       }
       png(sprintf("plots/part1/question2/q2g_parte_hist_parte_n%d_m%d.
↪png",n,m))
```

```
        h = hist(degree_neighbors, breaks=seq(0, max(degree_neighbors)),␣
↪freq=FALSE, main ="Histogram of degree distribution", xlab="Degree")
        dev.off()
        plot_x = tail(h$breaks, length(h$breaks) - 1)
        plot_y = h$density
        idx = which(plot_y != 0, arr.ind=TRUE)
        log_degree = log(seq(1:length(plot_y)))[idx]
        log_distribution = log(plot_y)[idx]
        relation = lm(log_distribution ~ log_degree)
        cat("\n")
        print(relation)
        png(sprintf("plots/part1/question2/q2g_parte_logplot_n%d_m%d.png",n,m))
        plot(log_degree, log_distribution,abline(relation,col="red"),␣
↪main=sprintf("Degree distribution of the jth node for n=%d m=%d (log-log␣
↪plot)",n,m), xlab="log(Degree)", ylab="log(Density)")
        dev.off()
    }

    print("2f")
    n = 1000
    cat(paste("\nn= ", n))
    cat(paste("\nm= ", m))
    iteration_times =1000
    degrees.sum = vector(mode = "numeric", length = iteration_times)
    for (i in 1:iteration_times){
        g = barabasi.game(n=n,m = m, directed=F)
        degrees.sum = degrees.sum + degree(g)
    }
    png(sprintf("plots/part1/question2/q2g_partf_age_m%d.png",m))
    plot(seq(1:iteration_times),rev(degrees.sum/iteration_times)␣
↪,main=sprintf("Age of Nodes vs Expected Degree (n=%d m=%d)",n,m),xlab="Age␣
↪of Nodes",ylab="Expected Degree",type="l")
    dev.off()
}
```

```
[1] "2a, 2b, 2c"

n=  1000
m=  2
[1] "Is the graph always connected : TRUE"

Modularity is  0.527829425660773
n=  10000
m=  2
[1] "Is the graph always connected : TRUE"

Modularity is  0.530985359910811[1] "2d"
```

```
n=  1000
m=  2

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
      1.209       -2.354


n=  10000
m=  2

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
      1.019       -2.452

[1] "2e"

n=  1000
m=  2

Call:
lm(formula = log_distribution ~ log_degree)

Coefficients:
(Intercept)   log_degree
    -0.9971      -1.1291


n=  10000
m=  2

Call:
lm(formula = log_distribution ~ log_degree)

Coefficients:
(Intercept)   log_degree
    -0.8447      -1.3433

[1] "2f"

n=  1000
```

```
m=  2[1] "2a, 2b, 2c"

n=  1000
m=  5
[1] "Is the graph always connected : TRUE"

Modularity is  0.281452119648816
n=  10000
m=  5
[1] "Is the graph always connected : TRUE"

Modularity is  0.27887342555672[1] "2d"

n=  1000
m=  5

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
      1.243       -1.972


n=  10000
m=  5

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
      1.371       -2.195

[1] "2e"

n=  1000
m=  5

Call:
lm(formula = log_distribution ~ log_degree)

Coefficients:
(Intercept)   log_degree
    -1.1439      -0.9575


n=  10000
```

```
m=  5

Call:
lm(formula = log_distribution ~ log_degree)

Coefficients:
(Intercept)   log_degree
     -0.935       -1.205

[1] "2f"

n=  1000
m=  5
```

[16]:
```r
#2h) Stub matching

n = 1000
m = 1
g1 = barabasi.game(n=n, m=m, directed=FALSE)
degree1 = degree(g1)
g_comm1 = fastgreedy.community(g1)
modularity_g1 = modularity(g_comm1)
cat(paste("\nModularity of Original Network is ", modularity_g1))

png(sprintf("plots/part1/question2/q2h_original_network.png"))
plot(g1,vertex.size=2, vertex.label=NA,main="Original undirected Network with␣
 ↪Preferential Attachment")
dev.off()

png(sprintf("plots/part1/question2/q2h_original_commstruct.png"))
plot(g_comm1, g1, vertex.size=2, vertex.label=NA,main="Community Structure of␣
 ↪Original Network")
dev.off()

g2 = sample_degseq(degree1, method="simple.no.multiple") # Generate graph with␣
 ↪same degree sequence
g_comm2 = fastgreedy.community(g2)
modularity_g2 = modularity(g_comm2)
cat(paste("\nModularity of New Network is ", modularity_g2))

png(sprintf("plots/part1/question2/q2h_new_network.png"))
plot(g2,vertex.size=2, vertex.label=NA,main="New network with the same Degree␣
 ↪Sequence")
dev.off()

png(sprintf("plots/part1/question2/q2h_new_commstruct.png"))
```

```
plot(g_comm2, g2, vertex.size=2, vertex.label=NA,main="Community Structure of␣
 ↪new Network")
dev.off()
```

```
Modularity of Original Network is  0.93380968556144
```

png: 2

png: 2

```
Modularity of New Network is  0.832903474044617
```

png: 2

png: 2