Homework 1 - Part A

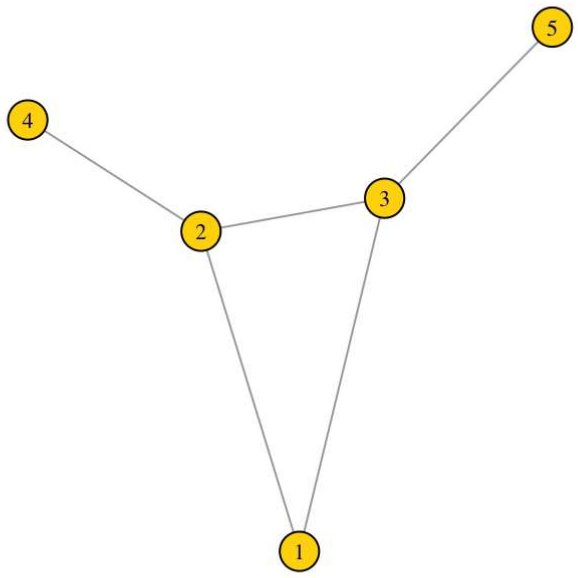
Leonardo Masci and Tansel Simsek (Group 20)

In this exercise, we are asked to implement a randomized max-cut algorithm and to analyze its performance by comparing its results with those of a given function (`maxcut`), which gives the optimal solution of the Max-Cut problem. In order to achieve this, we will first define a small graph, find its max-cut via the two methods mentioned above and compare the results. Then, we will move on to a bigger graph, to check the impact of the graph's size on the two methods.

Firstly, we prepare the environment, by requiring all the packages we will need.

Now, we want to set up a simulation to analyze the effectiveness of the randomized max-cut algorithm. In order to achieve this, we start off with a notable small graph: using the function `make_graph` from the package `igraph` , we create a "Bull" graph, which has 5 vertices and 5 edges.

```
bull<-make_graph("Bull")
plot(bull,vertex.color="gold",vertex.label.cex=0.7)
```



Next, we compute the max-cut by using the `maxcut` function from the `sdpt3r` package. In order to take advantage of this function, we first need to convert our plot into a matrix. After that, we are able to apply the function and obtain the maximum cut of the graph, whose size constitutes our optimal value. The maximum cut will be a negative number, but we turn it into a positive number since we are interested in its size.

```
matrix_bull <-as.matrix(as_adjacency_matrix(bull, type = c("both", "upper", "lower"),
                        attr = NULL, edges = FALSE, names = TRUE,
                        sparse = igraph_opt("sparsematrices")))

out<-maxcut(matrix_bull)
opt<-(out$pobj)*(-1)
opt
```

```
## [1] 4.25
```

We get a Opt of 4.25 for the Bull graph.

Now, we implement our own algorithm, being the randomized max-cut algorithm. Our goal is to see the average cut we can obtain in this way, therefore we run it 1000 times and take the average. In the algorithm we flip a coin for each vertex: we add said vertex to the subset U if we get tails and discard it if we get heads. We then extract the size of the cut obtained from removing from the original Bull graph all the vertices not found in U, and finally calculate the mean of such cuts.

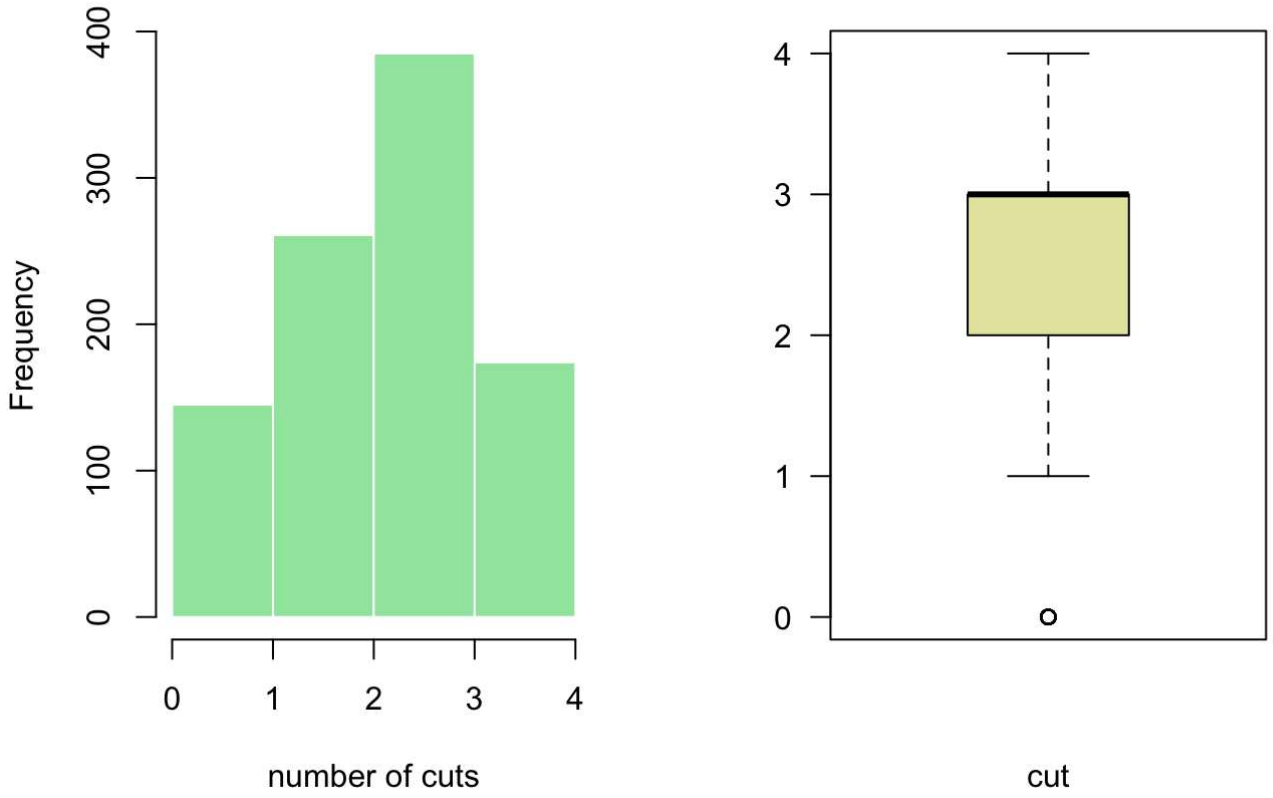
```
#preliminary variables we will need
avr_cuts<-c()
v<-length(V(bull))
V(bull)$name <- as.character(1:v)

#we run the algorithm 1000 times
average_cuts=0
M=1000

#the randomized max-cut algorithm
for (x in 1:M){
  coin<-c()
  for(u in 1:v){
    coin=sample(c(0,1),v,replace=TRUE,prob=c(.5,.5))
  }
  U<-c()
  notU<-c()
  for (i in c(1:v)){
    if(coin[i]==1){
      U<-c(U,i)
    }
    if(coin[i]==0){
      notU<-c(notU,i)
    }
  }
  if (length(U)>0){
    bulltest<-subgraph.edges(bull,E(bull)[inc(V(bull)[U])])
    for (m in U){
      for (j in U){
        if (are.connected(bulltest, V(bulltest)$name==m ,V(bulltest)$name==j)){
          bulltest <- delete.edges(bulltest,E(bulltest,P=c(which(V(bulltest)$name==m),
                                                           which(V(bulltest)$name==j))))
        }
      }
    }
    average_cuts =  average_cuts + gsize(bulltest)
    avr_cuts <- c(avr_cuts,gsize(bulltest))
  }
}
average_cuts = average_cuts/M
average_cuts
```

```
## [1] 2.476
```

So the average cut is equal to 2.5.

After running the algorithm, we now plot the distribution of the number of cuts obtained within the simulation.

```
nf <- layout( matrix(c(1,2), ncol=2) )
hist(avr_cuts , breaks=4 , border=F , col=rgb(0.1,0.8,0.3,0.5) , xlab="number of cuts" , main
="")
boxplot(avr_cuts , xlab="cut" , col=rgb(0.8,0.8,0.3,0.5) , las=2)
```



From the plots above we can see that in our final output we do not take into consideration the times when no vertices are cut off or when all of them are, since none of those constitute an actual cut. Instead, we observe that the most common cut is between 2 and 3, which is also equal to the average cut. This is because the distribution of the number of cuts in the simulation is a normal distribution. This is more clear in the next example, where the number of cuts is bigger.

Lastly, we compare this average cut with Opt/2.

```
print(paste0("Average cut-size of the bull graph is ", as.character(average_cuts),"."))
```

```
## [1] "Average cut-size of the bull graph is 2.476."
```

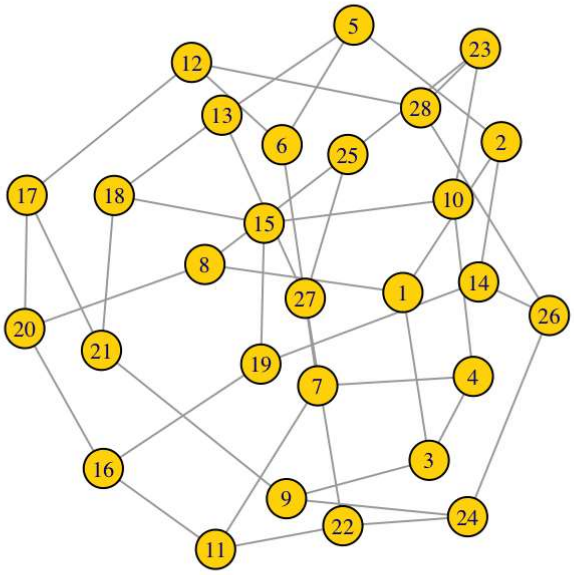
```
print(paste0("Half optimal cut value of the bull graph is ",as.character(abs(opt/2)),"."))
```

```
## [1] "Half optimal cut value of the bull graph is 2.12499999800534."
```

Therefore, the average cut obtained with the randomized algorithm is bigger than the theoretical bound Opt/2. We did obtain a satisfactory expected cut, so our algorithm seems to be very effective.

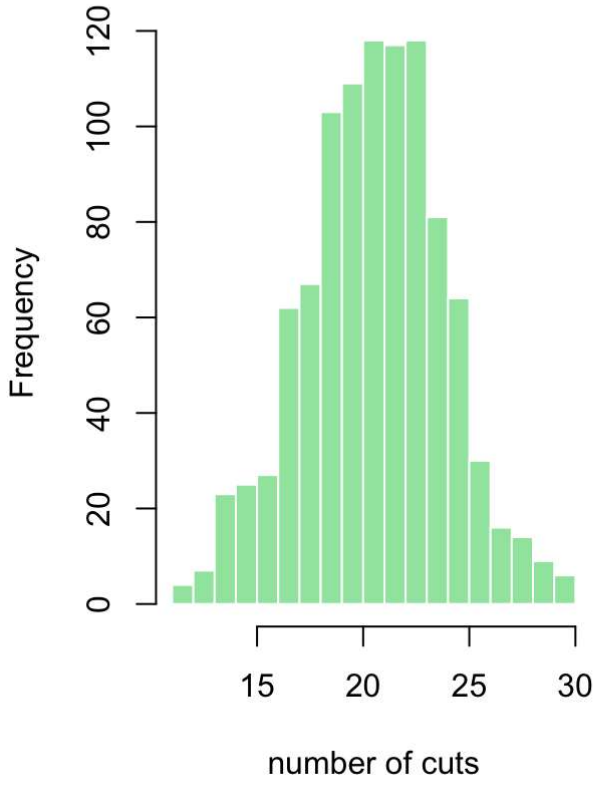
Before drawing any final conclusions, we want to repeat the experiment with a bigger graph, and analyze the new results. In order to achieve this, we selected the notable graph “Coxeter”, from the same `make_graph` function mentioned above. This graph has 28 vertices and 42 edges.

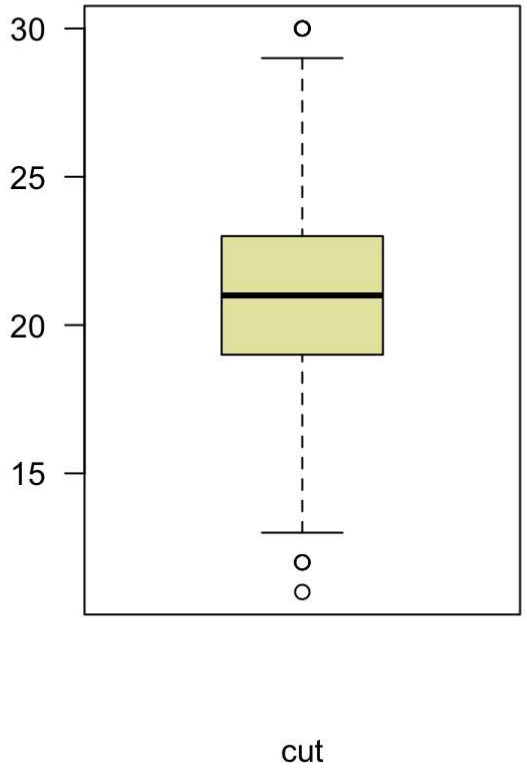
```
c<-make_graph("Coxeter")

plot(c,vertex.color="gold",vertex.label.cex=0.7)
```



Now, we compute the size of the cuts obtained via the two methods, again running the randomized algorithm 1000 times.

```
#preliminary variables needed
avr_cuts2<-c()
v<-length(V(c))
V(c)$name <- as.character(1:v)

#we run the algorithm 1000 times
average_cuts2=0
M=1000

#randomized algorithm
for (t in 1:M){
  coin<-c()
  for(u in 1:v){
    coin=sample(c(0,1),v,replace=TRUE,prob=c(.5,.5))
  }
  U<-c()
  notU<-c()
  for (i in c(1:v)){
    if(coin[i]==1){
      U<-c(U,i)
    }
    if(coin[i]==0){
      notU<-c(notU,i)
    }
  }
  if (length(U)>0){
    test<-subgraph.edges(c,E(c)[inc(V(c)[U])])
    for (m in U){
      for (j in U){
        if (are.connected(test, V(test)$name==m ,V(test)$name==j)){
          test <- delete.edges(test, E(test,P=c(which(V(test)$name==m),
                                                which(V(test)$name==j))))
        }
      }
    }
    average_cuts2 =  average_cuts2 + gsize(test)
    avr_cuts2 <- c(avr_cuts2,gsize(test))
  }
}

average_cuts2 = average_cuts2/M
nf <- layout( matrix(c(1,2), ncol=2) )
hist(avr_cuts2 , breaks=14 , border=F , col=rgb(0.1,0.8,0.3,0.5) , xlab="number of cuts" , ma
in="")
boxplot(avr_cuts2 , xlab="cut" , col=rgb(0.8,0.8,0.3,0.5) , las=2)
```






```
#maxcut function
B<-as.matrix(as_adjacency_matrix(c, type = c("both", "upper", "lower"),
                                 attr = NULL, edges = FALSE, names = TRUE,
                                 sparse = igraph_opt("sparsematrices")))
out<-maxcut(B)
opt2<-(out$pobj)*(-1)
```

We can see that the distribution of number of cuts among the simulations corresponds to a normal distribution, centering on the average cut.

```
#the two results
print(paste0("Average cut-size of the coxeter graph is ", as.character(average_cuts2),"."))
```

```
## [1] "Average cut-size of the coxeter graph is 21.019."
```

```
print(paste0("Half optimal cut value of the coxeter graph is ",as.character(abs(opt2/2)),"."
))
```

```
## [1] "Half optimal cut value of the coxeter graph is 18.9497473616727."
```

Once again, we find that the final average cut of our randomized graph is greater than that of the expected optimal value. To compare this result with the one found for the smaller graph, we compute the relative cuts.

```
print(average_cuts/(opt/2))
```

```
## [1] 1.165176
```

```
print(average_cuts2/(opt2/2))
```

```
## [1] 1.109197
```

We find that the size of the graph does have an impact on the randomized algorithm, in that it makes it less efficient, although it does remain bigger than the theoretical bound.

In conclusion, the expected size of the cut obtained with this randomized max-cut algorithm seems to be performing pretty well, consistently giving results higher than the designed limit value. The latter is defined as half of the optimal cut, which is found by applying a tailored function to the graph. Finally, our algorithm loses effectiveness in comparison to the max-cut function for bigger graphs.

Comment:
Ok, the results are consistents and the simulations are good, even if the code can be improved using some vectorization.

Homework 1 - Part B

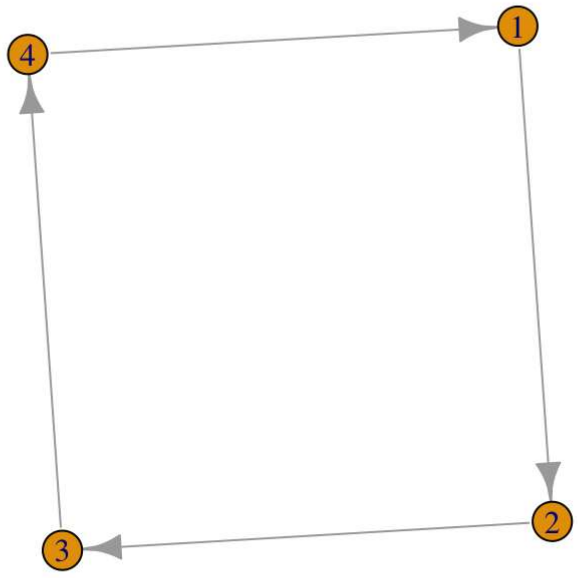
Leonardo Masci and Tansel Simsek (Group 20)

In this exercise, we are asked to simulate the preferential attachment process and to draw a plot showing its complimentary cumulative degree distribution.

Once again, we start off by loading the necessary packages into the library.

Before starting our simulations, we create the initial graph we will be working with. We used the ‘make_ring’ function from the igraph package to make a directed ring graph with 4 vertices that are labeled.

```
x<-make_ring(4,directed=TRUE)
plot(x)
```

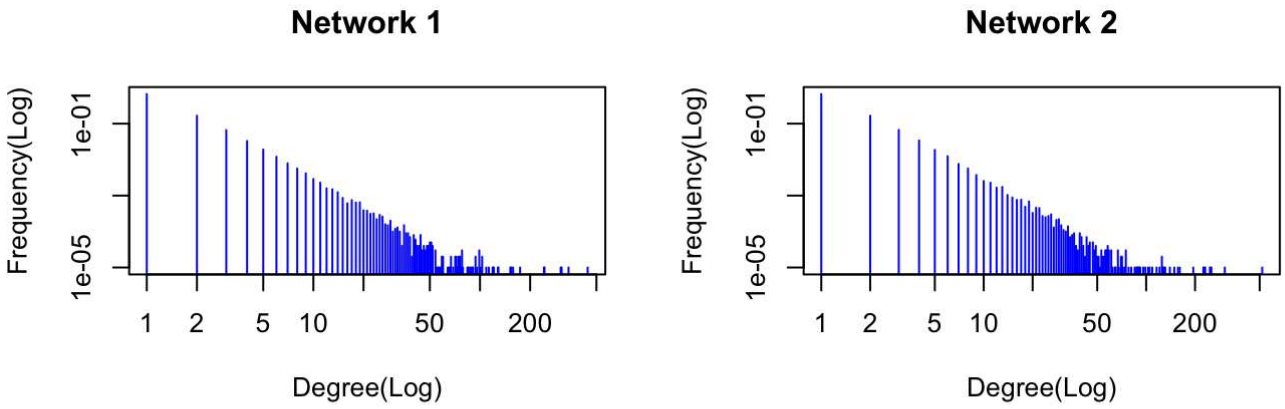


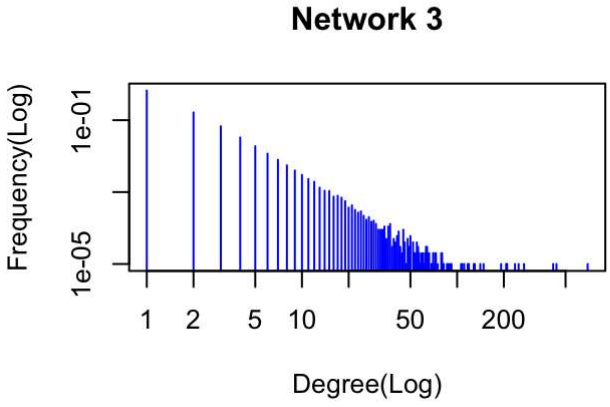
Next, we implement the preferential attachment process for 4 networks, for each of which we continue adding pages until there are 100,000. The process starts off by flipping a coin for each vertex added, since the probability of choosing one method or the other is 0.5. When the coin lands on heads (t=0), the new page linked to another chosen uniformly at random among the existing pages in the network. Simulating this in our graph was easy, as we simply had to uniformly extract a vertex among the vertices in the graph and link the new vertex to that one. On the other hand when the coin lands on heads (t=1) the new page’s link is copied from existing links. To achieve this, we created an array containing every vertex that has another vertex linked to it. For example, if the vertex 1 had three links it would appear three times in the array. In this way, we were able to obtain weighted probabilities by uniformly sampling from that array.

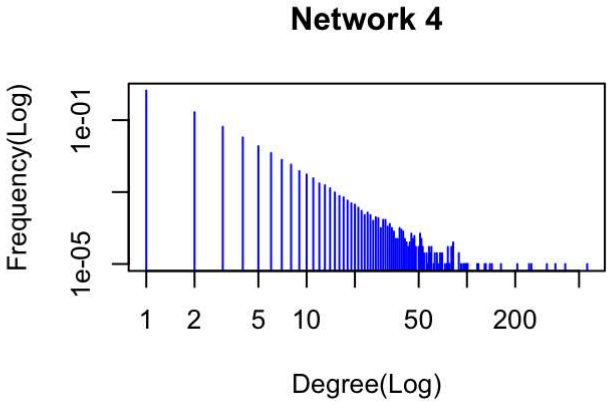
```
network_list <- list()
for (j in 1:4){
  x<-make_ring(4,directed=TRUE)
  g<-.5
  v<-length(V(x))
  a<-as.integer(V(x))
  V(x)$name <- as.character(1:v)
  for (i in 5:100000){
    x<-add.vertices(x,1,attr=c(name=i))
    t<-sample(c(0,1),1,replace=TRUE,prob=c(g,1-g))
    if (t==0){
      v<-length(V(x))
      V(x)$name <- as.character(1:v)
      which_vertex<-as.integer(runif(1,min=1,max=v))
      a<-c(a,which_vertex)
      x<-add.edges(x,c(i,which_vertex))
    } else {
      which_vertex<-sample(a,1,replace=TRUE,prob=c(rep(1/length(a),length(a))))
      a<-c(a,which_vertex)
      x<-add.edges(x,c(i,which_vertex))
    }
  }
  network_list <- append(network_list, list(x))
  nam <- paste("Network ", j, sep = "")
  assign(nam, x)
  names <- c(names, nam)
}
```

After creating our networks, we draw the log-log plots of their empirical degree distribution, which show the number of vertices that correspond to each degree. The degree of a vertex is the number of links to other vertices it has.

```
par(mfrow=c(2,2))

for (i in 1:length(network_list)){
dd <- degree_distribution(network_list[[i]])
plot(dd[2:length(dd)], log="xy", main = names[i+1],xlab = "Degree(Log)", ylab = "Frequency(Lo
g)", col = "blue", type = "h")
}
```



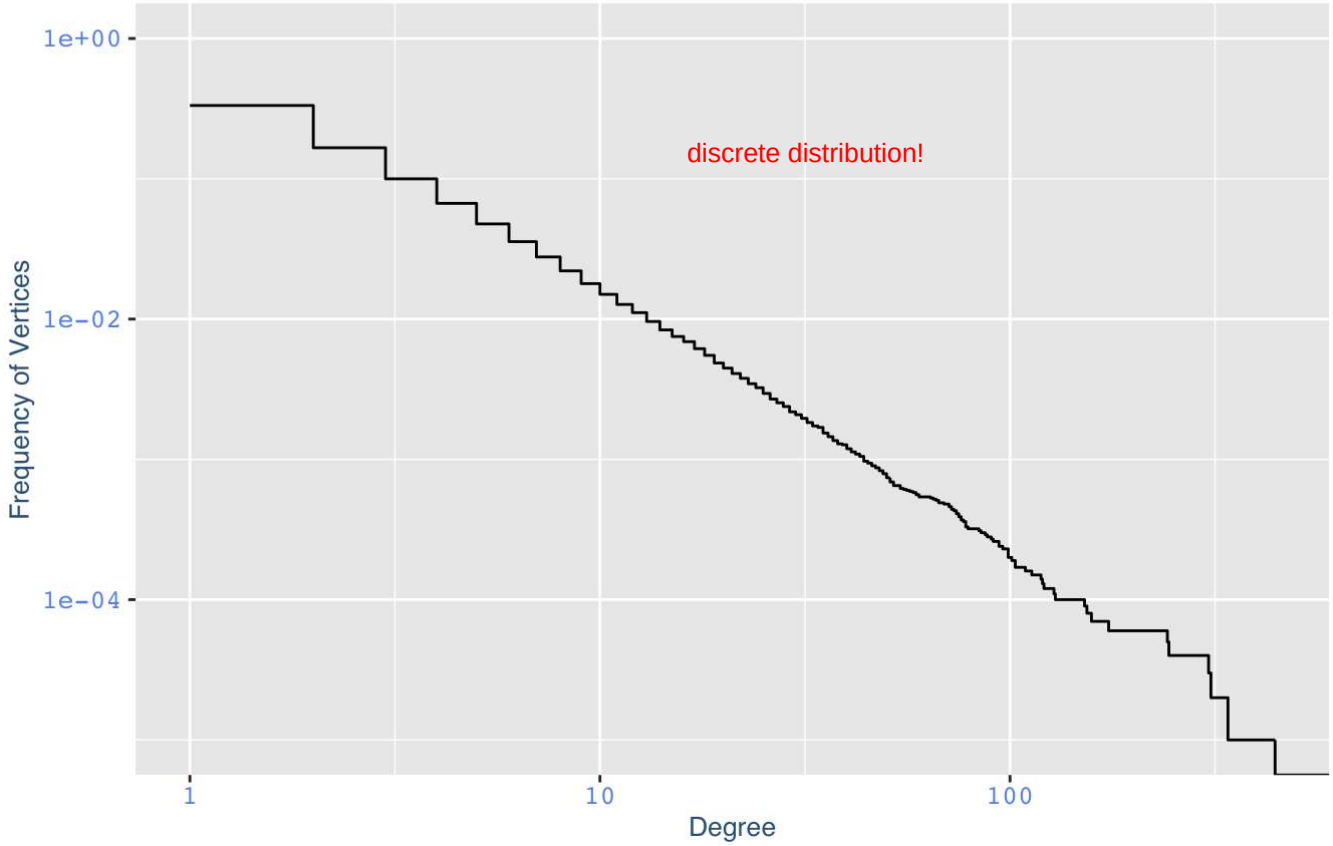




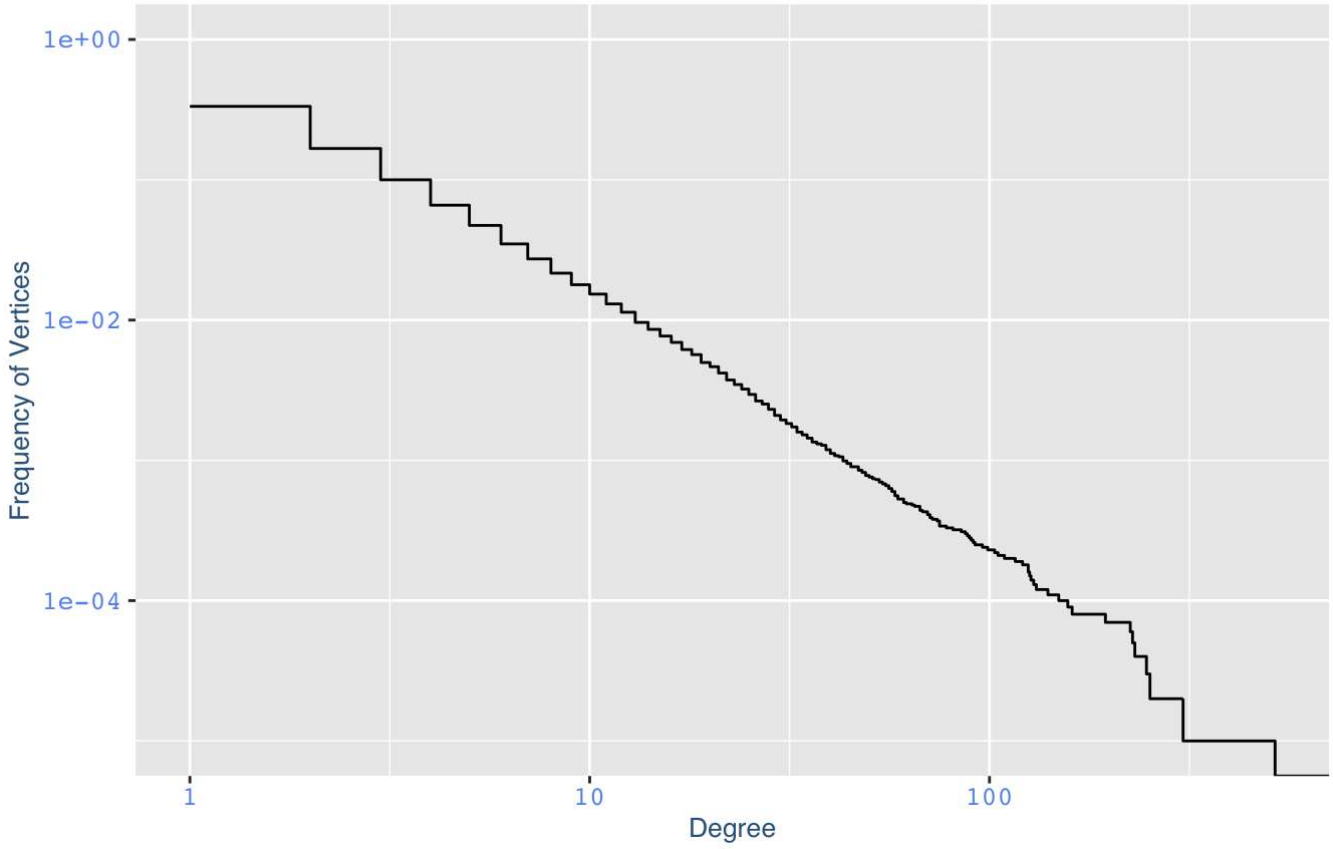
Finally, we show the log-log plots of the complimentary cumulative degree distributions.

```
mynamestheme <- theme(plot.title = element_text(family = "Helvetica", face = "bold", size = (
15)),
                  legend.title = element_text(colour = "steelblue",  face = "bold.italic", fam
ily = "Helvetica"),
                  legend.text = element_text(face = "italic", colour="steelblue4",family = "He
lvetica"),
                   axis.title = element_text(family = "Helvetica", size = (10), colour = "stee
lblue4"),
                   axis.text = element_text(family = "Courier", colour = "cornflowerblue", siz
e = (10)))
for (i in 1:length(network_list)){
  df <- degree(network_list[[i]])
  df <- data.frame(df)
  df <- data.frame(x = df$df)
  p  <- ggplot(df, aes(x)) + stat_ecdf()
  pg <- ggplot_build(p)$data[[1]]
  print(ggplot(pg, aes(x = x, y = 1-y ))+mynamestheme +labs(title = paste(names[i+1], "\nEmpi
rical Degree Distribution(log-log)"), y="Frequency of Vertices", x = "Degree") + geom_step()
 +scale_x_log10() +scale_y_log10())
}
```

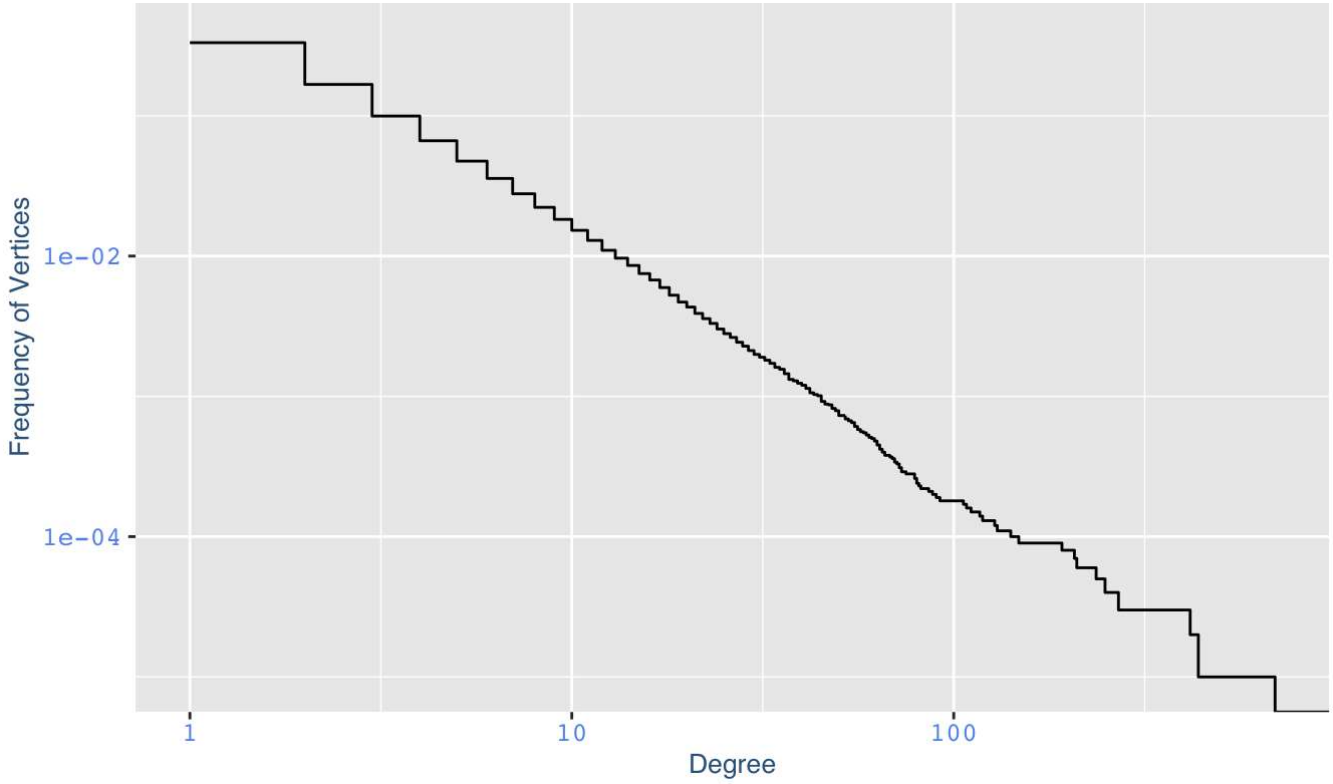
Network 1
Empirical Degree Distribution(log-log)



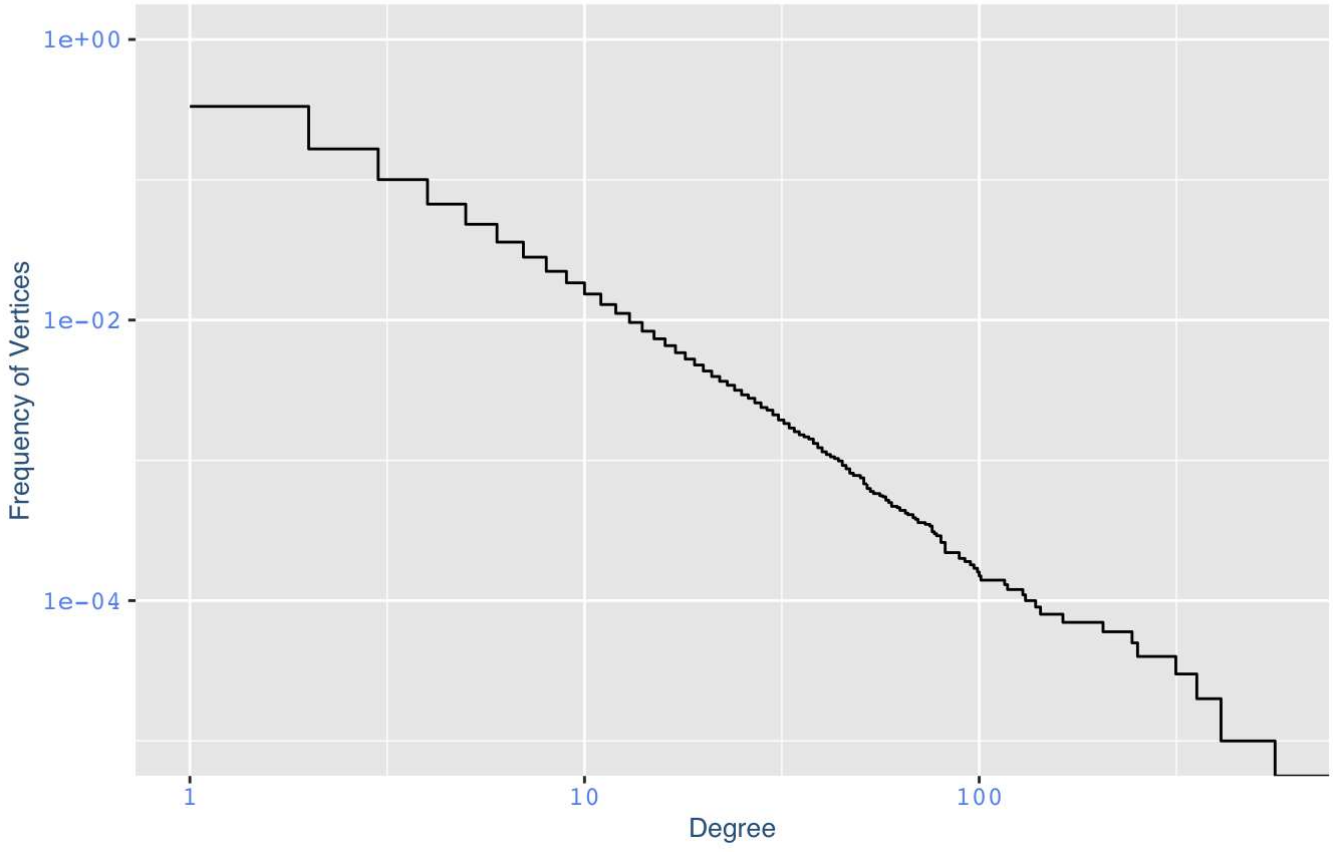
Network 2
Empirical Degree Distribution(log-log)



Network 3
Empirical Degree Distribution(log-log)

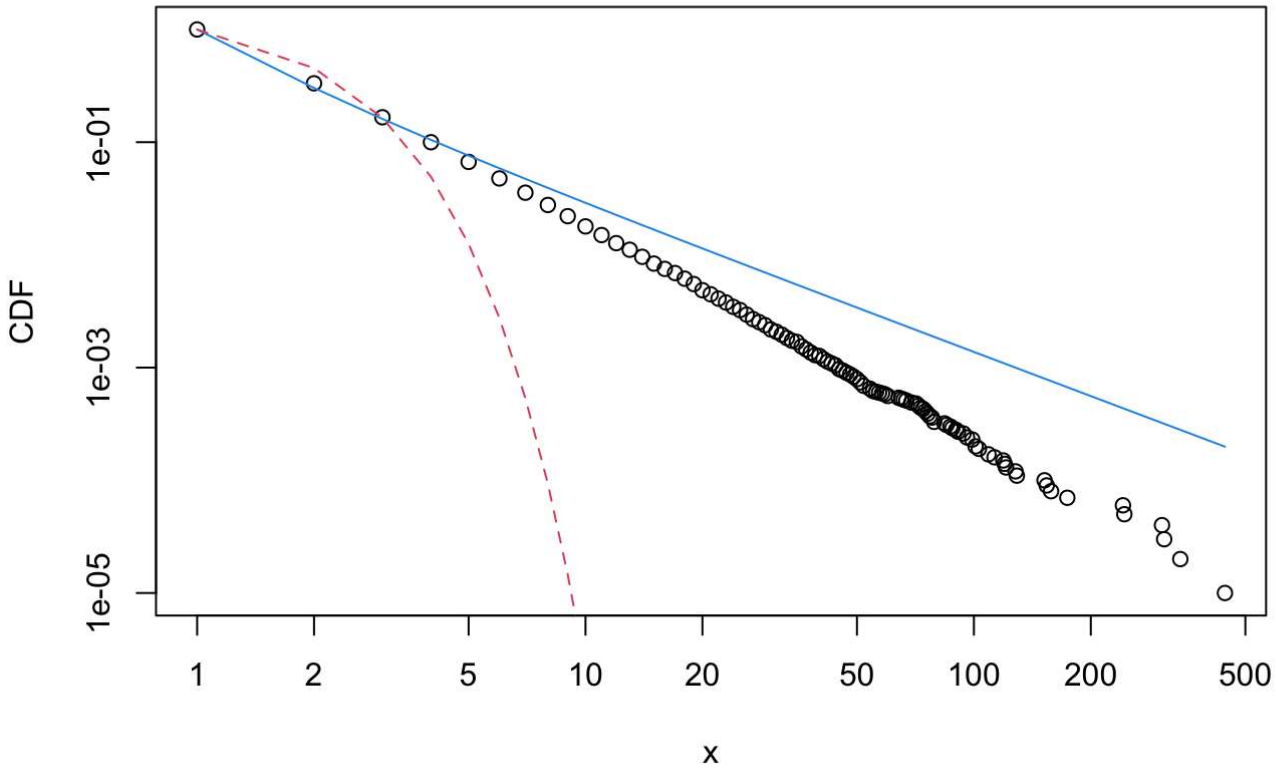


Network 4
Empirical Degree Distribution(log-log)



We now want to check whether the degree distributions follow a Poisson or a power law. Since all of the networks have similar distributions, we can procede with our analysis using just one of them. We chose the first network. In order to do this, we will be using the package `poweRlaw` and some of its functions. This way we are able to create a plot showing our distribution and both models, to compare how well they fit.

```
network<-network_list[[1]]
df <- degree(network)
df <- data.frame(df)
df <- data.frame(x = df$df)
values <- df$x
set.seed(1)
m1 = displ$new(values)
m1$setPars(estimate_pars(m1))
m2 = dispois$new(values)
m2$setPars(estimate_pars(m2))
plot(m2, ylab = "CDF")
lines(m1, col = 4)
lines(m2, col = 2, lty = 2)
```



The red line being the Poisson distribution and the blue one the Power Law. It is clear that the latter is a better fit for our distribution, but we also want to prove this formally. Using the `compare_distributions` function we can check two hypothesis: 1. Both distributions are equally far from the true distribution 2. The second distribution is better than the first one

```
comp = compare_distributions(m1, m2)
comp$p_two_sided
```

```
## [1] 0
```

```
compare_distributions(m1, m2)$p_one_sided
```

```
## [1] 0
```

```
compare_distributions(m2, m1)$p_one_sided
```

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

For the first hypothesis, tested via `p_two_sided`, we get a value of 0.003, so we reject it. Therefore, one distribution must be better than the other one. To check which one is better we use `p_one_sided`. As we would expect, we get opposite values for the two hypotheses, rejecting the first one. So we can conclude that m1 is a better fit than m2. In our case, m1 is the power law distribution, which makes sense in light of what we saw in the graph above.

In conclusion, the degree distribution seems to be following a power law distribution.

Comment:

The simulations are good, even if the code could be improved. I would have appreciated some additional comments and observations.