Statistical Analysis of Networks

R Markdown

Check graph

You are not required to write a *markdown document* containing text and code. If you do want to use it, create an `Rmd` file ans then use RStudio to convert to `pdf`. Please upload `pdf` file to Gradescope.

You will need the **networkdata** and **sna** packages. To install the packages from inside **R**:

```
install.packages("network")
install.packages("sna")
install.packages("networkdata",repos="http://www.stat.ucla.edu/~handcock")

options(tinytex.verbose = TRUE)

LC_ALL="en_US.UTF-8"
LC_CTYPE="en_US.UTF-8"
LANGUAGE="en_US.UTF-8"
```

1. *Centrality:* Here we consider the three network data sets in the **networkdata** package

```
# {r gapminder, echo=FALSE}
library(networkdata)

## Loading required package: statnet.common

## Warning: package 'statnet.common' was built under R version 3.6.2

##
## Attaching package: 'statnet.common'

## The following object is masked from 'package:base':
##
##     order

## Loading required package: network

## Warning: package 'network' was built under R version 3.6.2

## network: Classes for Relational Data
## Version 1.16.1 created on 2020-10-06.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##                     Mark S. Handcock, University of California -- Los
Angeles
##                     David R. Hunter, Penn State University
##                     Martina Morris, University of Washington
##                     Skye Bender-deMoll, University of Washington
```

```
##  For citation information, type citation("network").
##  Type help("network-package") to get started.

##
## networkdata: version 0.6, created on 2017-11-23
## Copyright (c) 2017, Mark S. Handcock, University of California -- Los
Angeles
##                     with contributions from
##                     Peter Hoff
##                     Vladimir Batagelj
##                     David M. Schruth, University of Washington
##                     Carter T. Butts, University of California -- Irvine
## Based on "statnet" project software (statnet.org).
## For license and citation information see statnet.org/attribution
## or type citation("networkdata").

data(butland_ppi)
data(addhealth9)
data(tribes)
```

Useful other packages are:

```
library(sna)

## Warning: package 'sna' was built under R version 3.6.2

## sna: Tools for Social Network Analysis
## Version 2.6 created on 2020-10-5.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##  For citation information, type citation("sna").
##  Type help(package="sna") to get started.

library(network)
```

You can check the nature of the networks, via, **help(tribes)**, etc. These networks need to be processed a bit for the analysis to match those in the . For the Add Health 9 network, consider only the (known) boys (via **addhealth9$X[,"female"]==0**). Next consider the undirected versions of the networks (For the Add Health 9 network we choose to say there is a tie if either boy nominates the other as a friend). Next consider only the largest component (see, e.g., **component.dist** in **sna**).

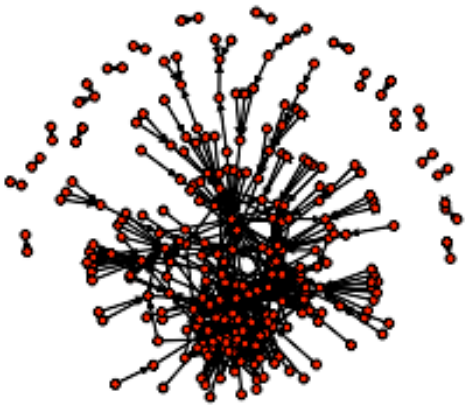
a) Create network objects from the networks, using either **network** or **igraph**.

```
tribes_pos = tribes[,,1]
tribes_neg = tribes[,,2]

# f_Data_to_Matrix <- function(data){
#   tribes_mat <- apply(as.matrix.noquote(data.frame(data)),2,as.numeric)
# }

f_Network <- function(data,dir,mat_type){
  net <- as.network(x = data, # the network object
```

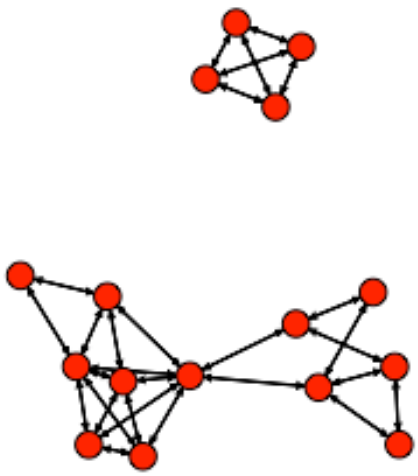
```
                  directed = dir, # specify whether the network is directed
                  loops = FALSE, # do we allow self ties (should not allow
them)
                  matrix.type = mat_type # the type of input
                     )
}

# Directed graphs
butland_ppi_net_dir = f_Network(butland_ppi,TRUE,"edgelist")
tribes_pos_net_dir = f_Network(tribes_pos,TRUE,"adjacency")
tribes_neg_net_dir = f_Network(tribes_neg,TRUE,"adjacency")
ind = which(addhealth9$X[,"female"]==0) # consider only the (known) boys
addhealth9_net_dir = f_Network(addhealth9$E[ind,],TRUE,"edgelist")

# Unirected graphs
butland_ppi_net_un = f_Network(butland_ppi,FALSE,"edgelist")
tribes_pos_net_un = f_Network(tribes_pos,FALSE,"adjacency")
tribes_neg_net_un = f_Network(tribes_neg,FALSE,"adjacency")
addhealth9_net_un = f_Network(addhealth9$E[ind,],FALSE,"edgelist")

library(sna)
# Largest components:
butland_ppi_cc = component.largest(butland_ppi_net_un,result = "graph")
tribes_pos_cc = component.largest(tribes_pos_net_un,result = "graph")
tribes_neg_cc = component.largest(tribes_neg_net_un,result = "graph")
addhealth9_cc = component.largest(addhealth9_net_un,result = "graph")

butland_ppi_cc_net = f_Network(butland_ppi_cc,TRUE,"adjacency")
tribes_pos_cc_net = f_Network(tribes_pos_cc,TRUE,"adjacency")
tribes_neg_cc_net = f_Network(tribes_neg_cc,TRUE,"adjacency")
addhealth9_cc_net = f_Network(addhealth9_cc,TRUE,"adjacency")
```

b) Plot the networks, as in the .

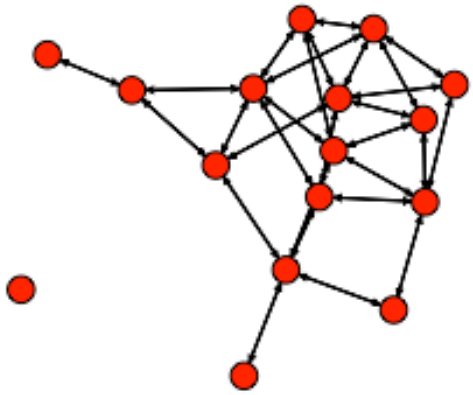
```
# Plot directed graphs
plot(butland_ppi_net_dir, vertex.cex = 1)
```



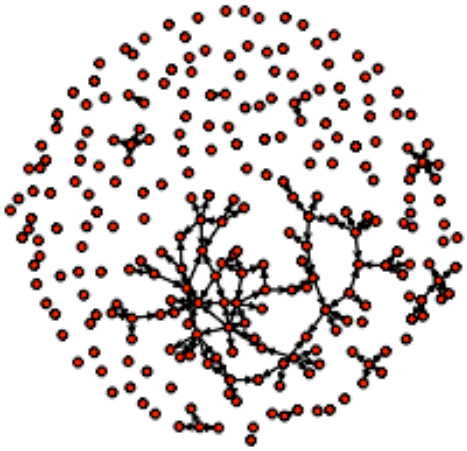
```
plot(tribes_pos_net_dir, vertex.cex = 3)
```



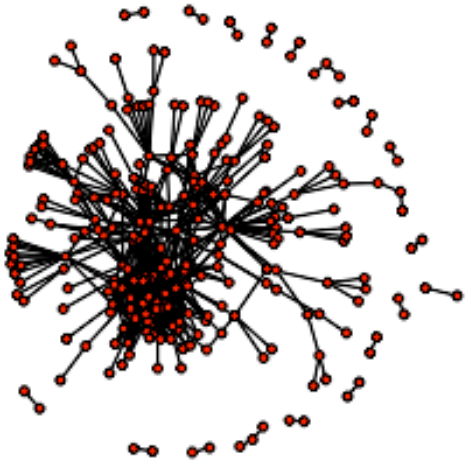
```
plot(tribes_neg_net_dir, vertex.cex = 3)
```



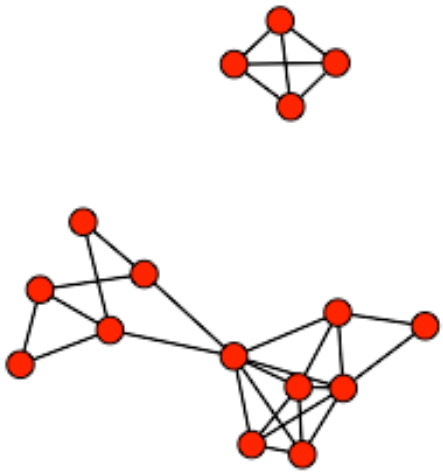
```
plot(addhealth9_net_dir, vertex.cex = 1)
```



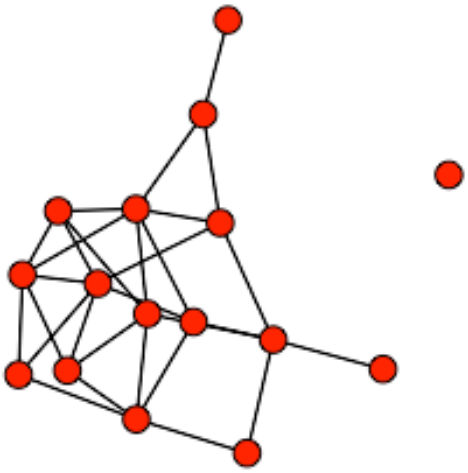
```r
# Plot undirected graphs
plot(butland_ppi_net_un, vertex.cex = 1)
```



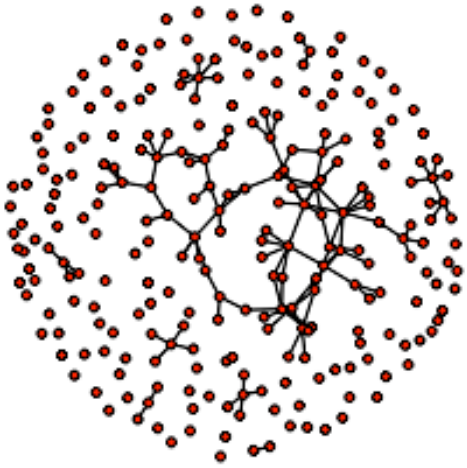
```
plot(tribes_pos_net_un, vertex.cex = 3)
```



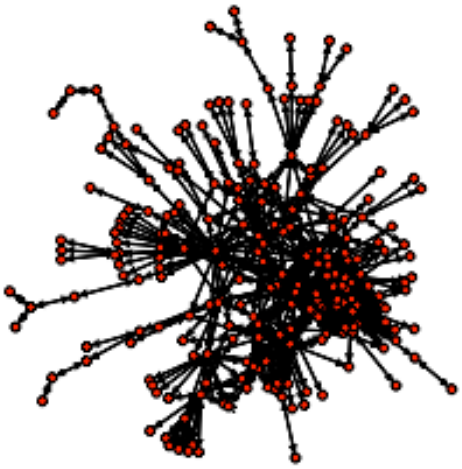
```
plot(tribes_neg_net_un, vertex.cex = 3)
```



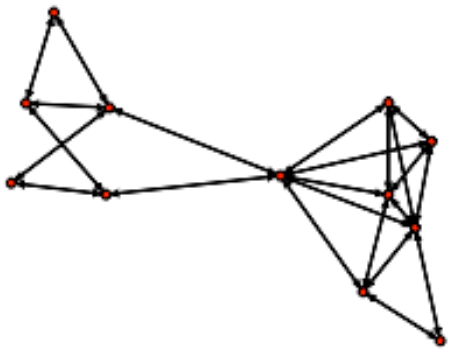
```
plot(addhealth9_net_un, vertex.cex = 1)
```



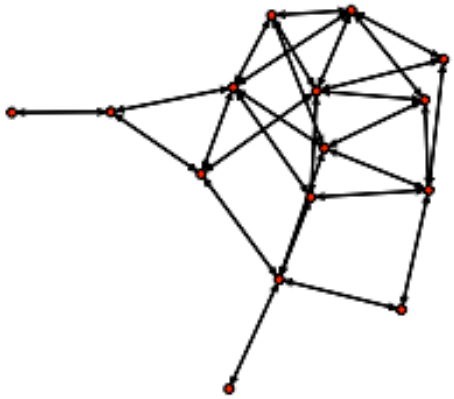
```r
# Plot the largest connected components
plot(butland_ppi_cc_net, vertex.cex = 1)
```



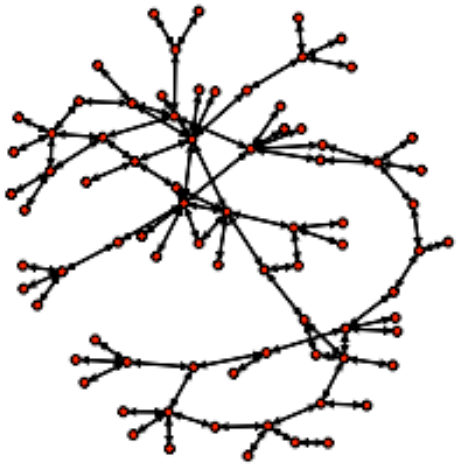
```r
plot(tribes_pos_cc_net, vertex.cex = 1)
```



```r
plot(tribes_neg_cc_net, vertex.cex = 1)
```



```r
plot(addhealth9_cc_net, vertex.cex = 1)
```

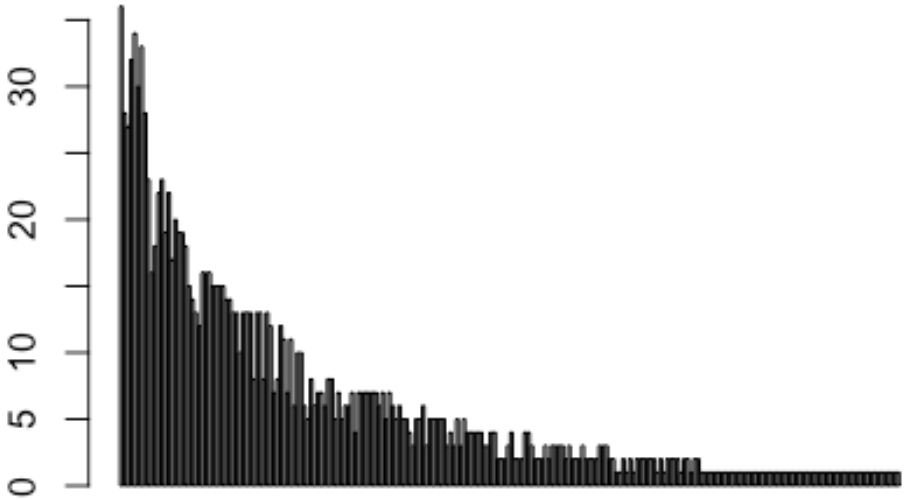


c) For each network, use the **degree()** function in the **sna** package to find the degree sequence. Note that for undirected networks, this function returns twice the degrees. You may also find the **table** command helpful. *Hint:* The **gmode** option in **degree** is important.

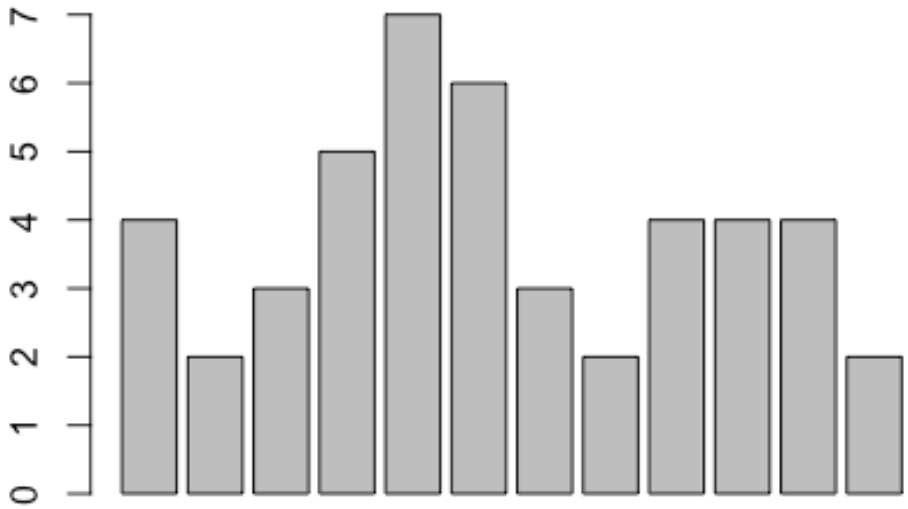
```
# Get degrees
butland_ppi_net_deg = degree(butland_ppi_cc_net,gmode="graph")
tribes_pos_net_deg = degree(tribes_pos_cc_net,gmode="graph")
addhealth9_net_deg = degree(addhealth9_cc_net,gmode="graph")
```

d) For each network, summarize the degree sequence using node-level graphical and numerical summaries (e.g., **barplot**).

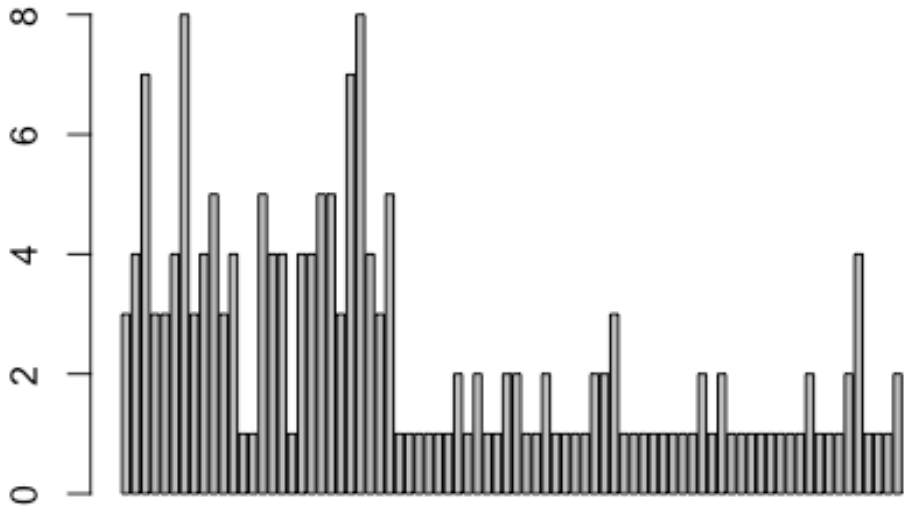
```
# Plot bar graphs of degrees
barplot(butland_ppi_net_deg)
```



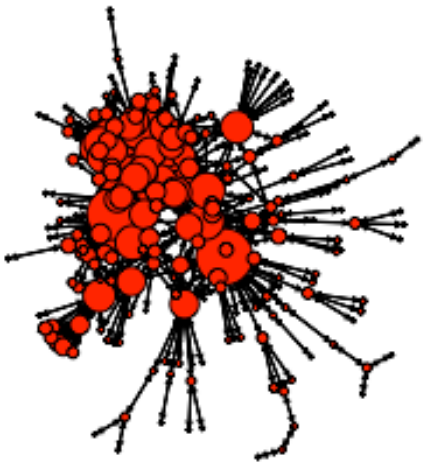

```
barplot(tribes_pos_net_deg)
```



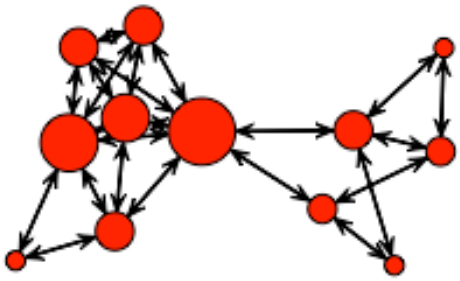

```
barplot(addhealth9_net_deg)
```



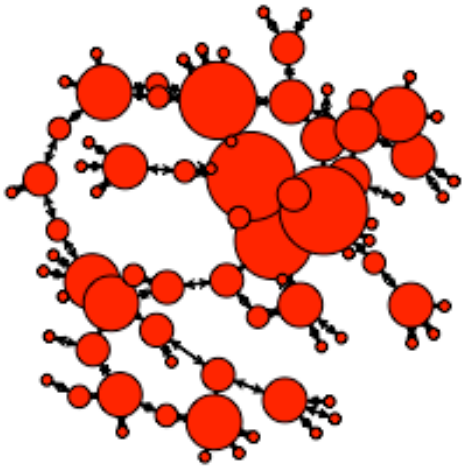

```
# Graphs with node sizes scaled by degrees
gplot(butland_ppi_cc_net, vertex.cex=(butland_ppi_net_deg)^0.8/2,
vertex.sides=50)
```



```
gplot(tribes_pos_cc_net, vertex.cex=(tribes_pos_net_deg)/2, vertex.sides=50)
```

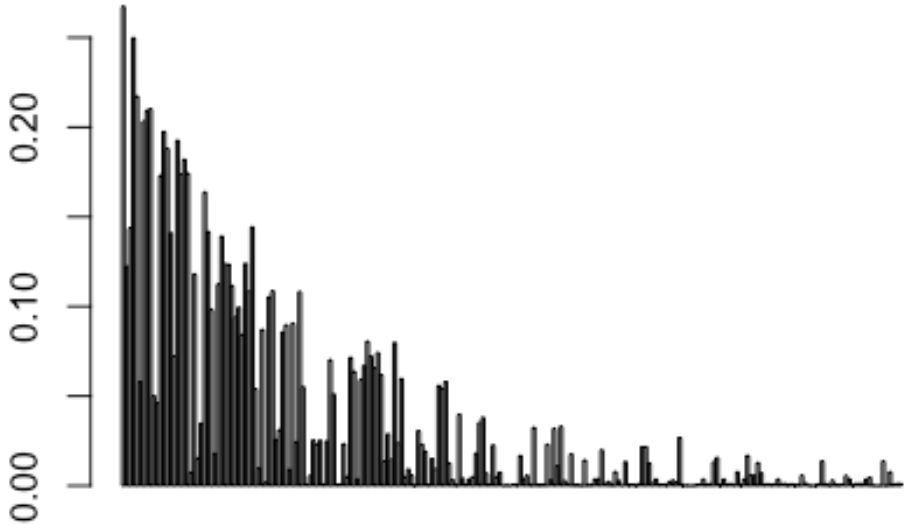


```
gplot(addhealth9_cc_net, vertex.cex=(addhealth9_net_deg), vertex.sides=50)
```

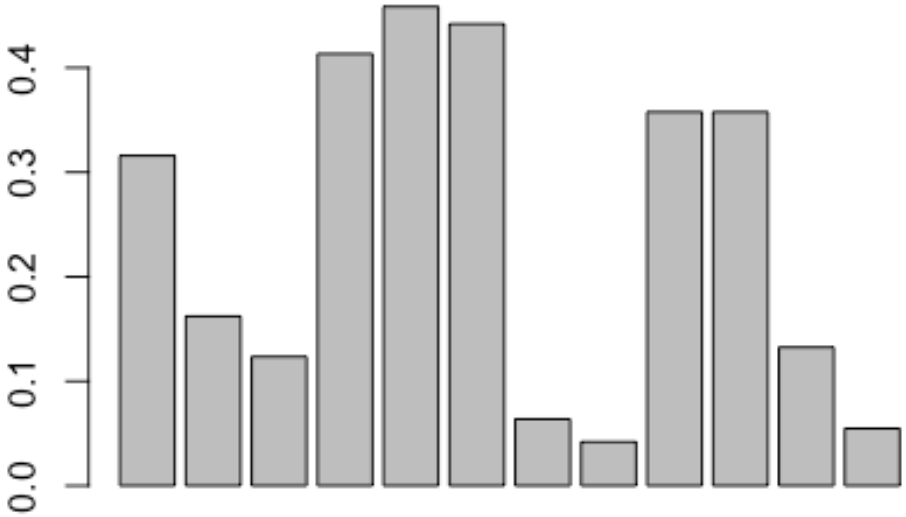


e) For each network, compute the eigenvalue centrality of each node (see **evcent** in **sna**). For each network, summarize the centralities using node-level graphical and numerical summaries (see **centrality** in **sna**). *Hint:* The **mode** option in **evcent** and **centralization** is important.

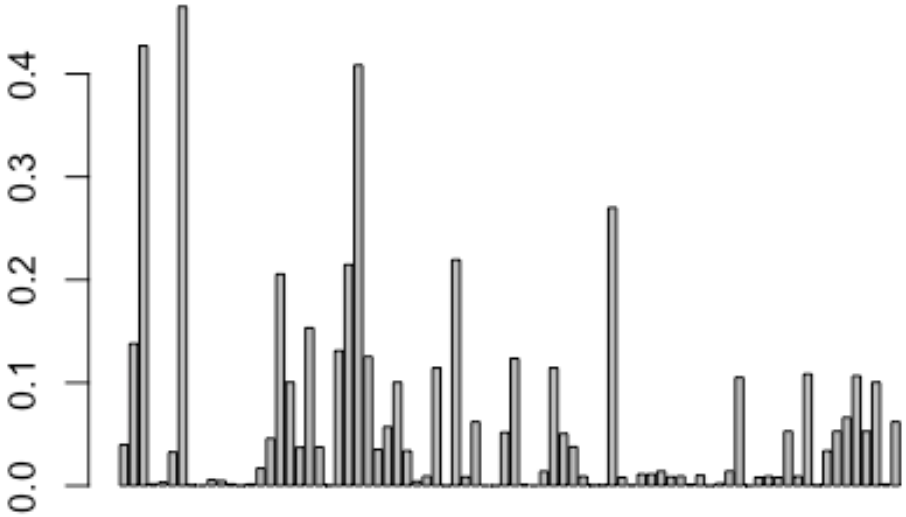
```
butland_ppi_net_ev = evcent(butland_ppi_cc_net, g=1, gmode="graph")
tribes_pos_net_ev = evcent(tribes_pos_cc_net, g=1, gmode="graph")
addhealth9_net_ev = evcent(addhealth9_cc_net, g=1, gmode="graph")

# Plot bar graphs of centrality
barplot(butland_ppi_net_ev)
```



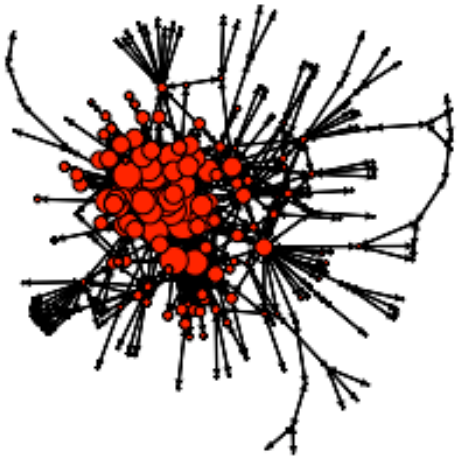

```
barplot(tribes_pos_net_ev)
```



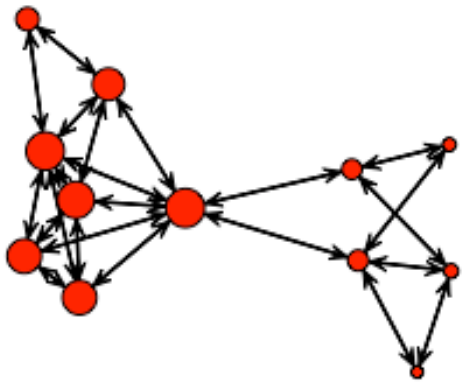

```
barplot(addhealth9_net_ev)
```



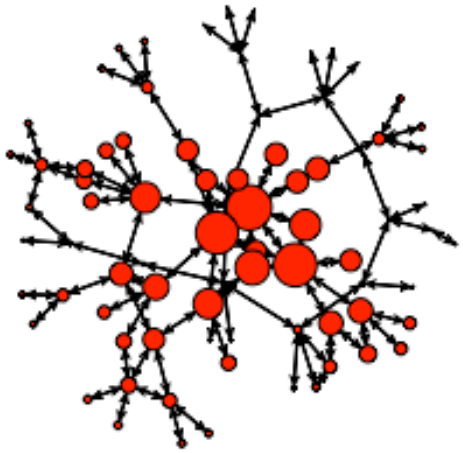

```
# Graphs with node sizes scaled by centrality
gplot(butland_ppi_cc_net, vertex.cex=(butland_ppi_net_ev)^0.5*10,
vertex.sides=50)
```



```
gplot(tribes_pos_cc_net, vertex.cex=(tribes_pos_net_ev)^0.5*3,
vertex.sides=50)
```

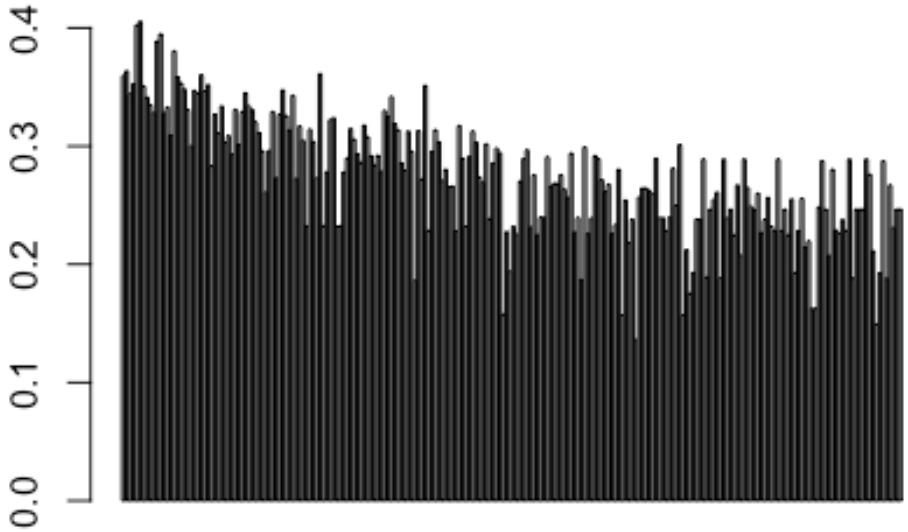


```
gplot(addhealth9_cc_net, vertex.cex=(addhealth9_net_ev)^0.5*6,
vertex.sides=50)
```

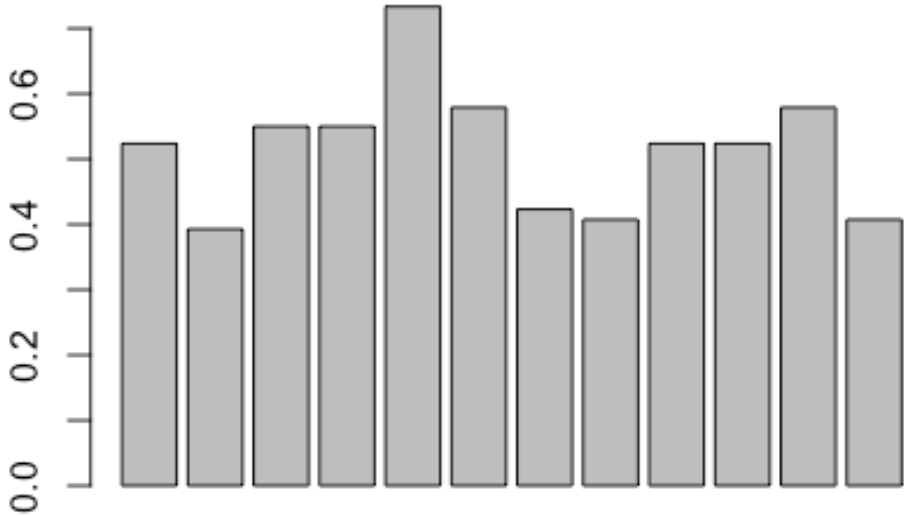


f) For each network, compute the closeness centrality of each node. For each network, summarize the centralities using node-level graphical and numerical summaries (see **closeness** in **sna**).

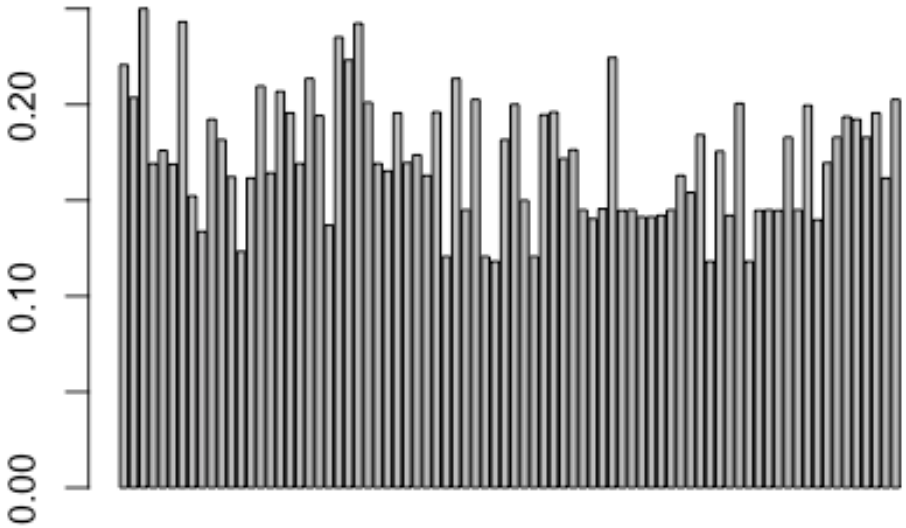
```
butland_ppi_closeness = closeness(butland_ppi_cc_net,g=1, gmode="graph")
tribes_pos_closeness = closeness(tribes_pos_cc_net,g=1, gmode="graph")
addhealth9_closeness = closeness(addhealth9_cc_net,g=1, gmode="graph")

# Plot bar graphs of closeness
barplot(butland_ppi_closeness)
```



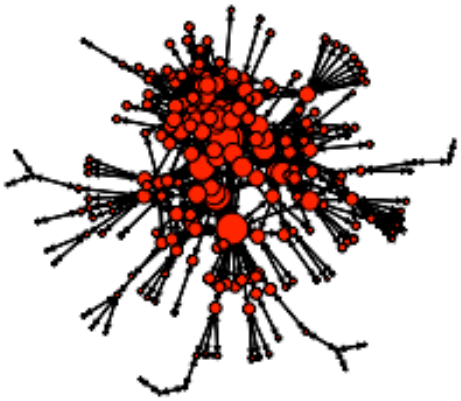

```
barplot(tribes_pos_closeness)
```



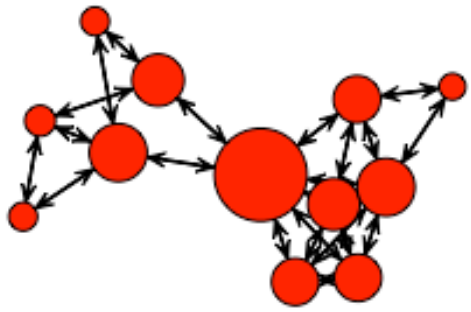

```
barplot(addhealth9_closeness)
```



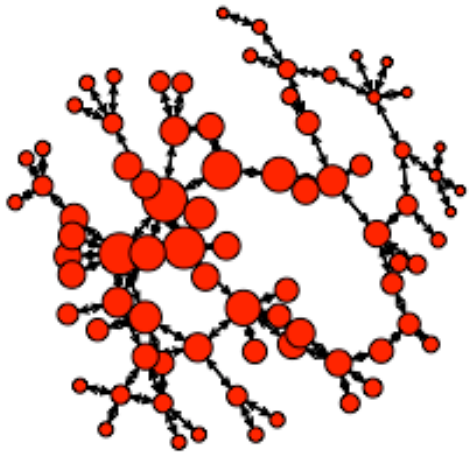

```
# Graphs with node sizes scaled by closeness
gplot(butland_ppi_cc_net, vertex.cex=((butland_ppi_closeness)*4)^3,
vertex.sides=50)
```



```
gplot(tribes_pos_cc_net, vertex.cex=((tribes_pos_closeness)*3)^2,
vertex.sides=50)
```

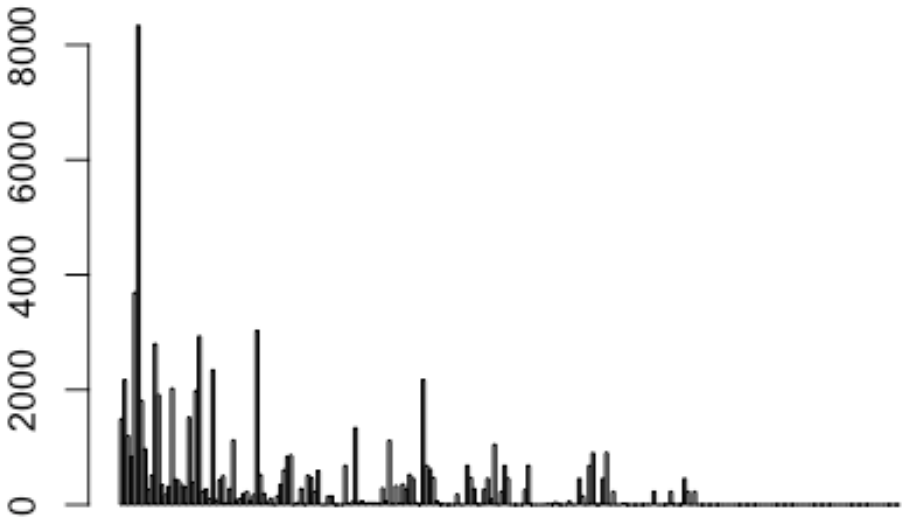


```
gplot(addhealth9_cc_net, vertex.cex=((addhealth9_closeness)*8)^2,
vertex.sides=50)
```

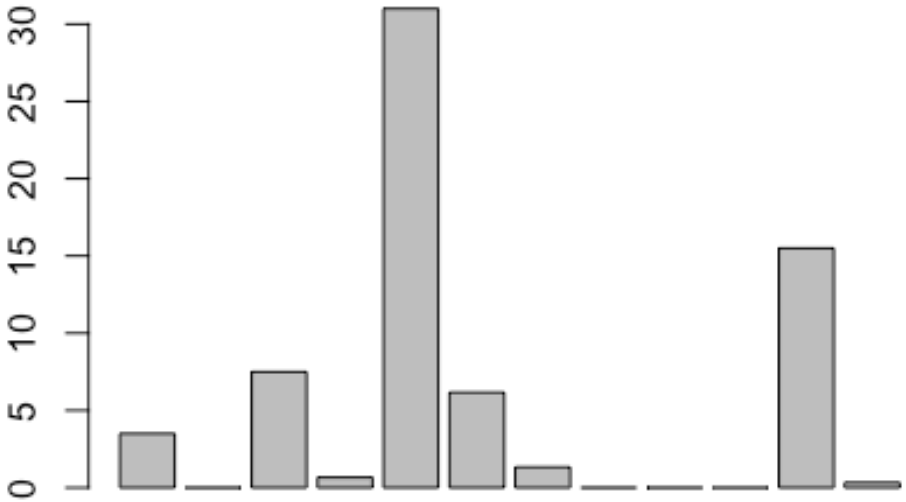


g) For each network, compute the betweenness centrality of each node. For each network, summarize the centralities using node-level graphical and numerical summaries (see **betweenness** in **sna**).

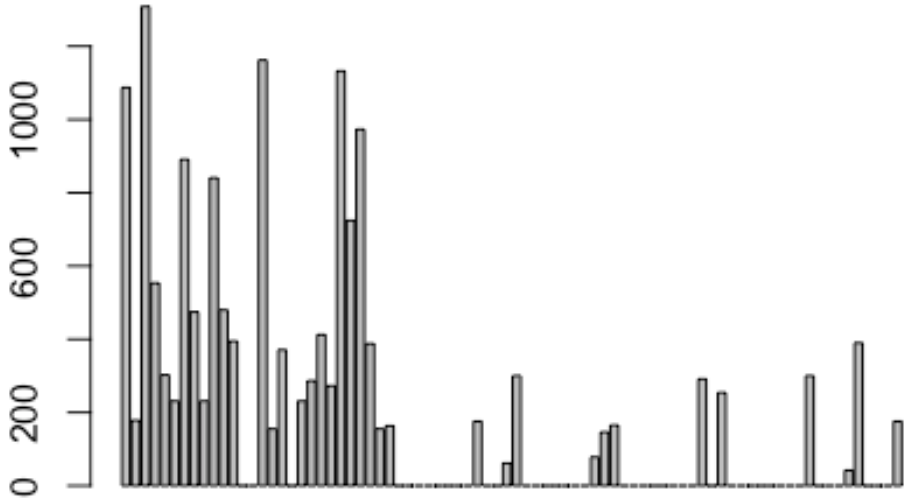
```
butland_ppi_betweenness = betweenness(butland_ppi_cc_net,g=1, gmode="graph")
tribes_pos_betweenness = betweenness(tribes_pos_cc_net,g=1, gmode="graph")
addhealth9_betweenness = betweenness(addhealth9_cc_net,g=1, gmode="graph")

# Plot bar graphs of betweenness
barplot(butland_ppi_betweenness)
```



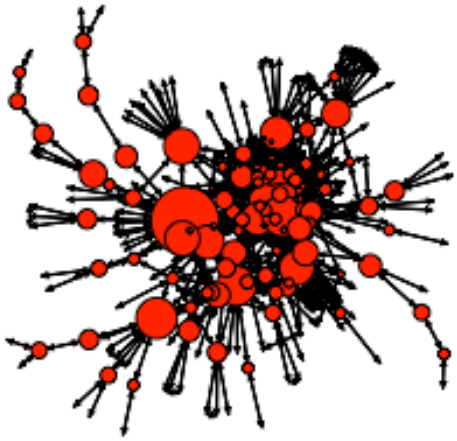

```
barplot(tribes_pos_betweenness)
```



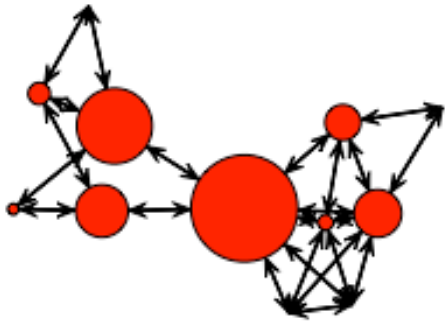

```
barplot(addhealth9_betweenness)
```



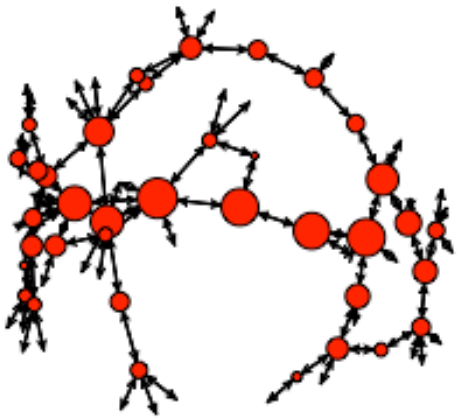

```
# Graphs with node sizes scaled by betweenness
gplot(butland_ppi_cc_net, vertex.cex=(butland_ppi_betweenness)^0.5/10,
vertex.sides=50)
```



```
gplot(tribes_pos_cc_net, vertex.cex=(tribes_pos_betweenness)^0.5,
vertex.sides=50)
```



```
gplot(addhealth9_cc_net, vertex.cex=(addhealth9_betweenness)^0.5/10,
vertex.sides=50)
```



h) For each network, compare the four centrality measures (degree, eigenvector, closeness and betweenness). Are they measuring the same thing?

Degree measures the number of connections to each node. In all three graphs, it identifies clusters of nodes. Eigenvector cenrality measures the limiting distibution of random walks on the graph. For these three graphs, the results look similar to degree centrality, as it seems to identify clusters of nodes. Betweenness measures the number of times a node lies on a shortest path between other nodes, and can therefore identify “bridges” in the network. This measure highly weights the nodes in the middle of the tribes data, where there is a clear bridge, and it also seems to heavily weight some nodes in the ppi data, which has a tree-like structure, but it is more spread out for the addhealth data, which contains cycles and is more connected. Closeness measures the sum of shortest paths from a node to all other nodes. It is thus a global measure of centrality. In these three graphs, it seems to give results which are similar to the betweenness centrality.

i) Reconstruct the network-level centralizations on the page ``Empirical study: Comparing centralization of different networks" of the .

```
ppi_deg = centralization(butland_ppi_cc_net, degree, mode="graph")
addhealth_deg = centralization(addhealth9_cc_net, degree, mode="graph")
```

```r
tribes_deg = centralization(tribes_pos_cc_net, degree, mode="graph")

ppi_close = centralization(butland_ppi_cc_net, closeness, mode="graph")
addhealth_close = centralization(addhealth9_cc_net, closeness, mode="graph")
tribes_close = centralization(tribes_pos_cc_net, closeness, mode="graph")

ppi_between = centralization(butland_ppi_cc_net, betweenness, mode="graph")
addhealth_between = centralization(addhealth9_cc_net, betweenness,
mode="graph")
tribes_between = centralization(tribes_pos_cc_net, betweenness, mode="graph")

ppi_ev = centralization(butland_ppi_cc_net, evcent, mode="graph")
addhealth_ev = centralization(addhealth9_cc_net, evcent, mode="graph")
tribes_ev = centralization(tribes_pos_cc_net, evcent, mode="graph")

print("Degree:")

## [1] "Degree:"

print(c("ppi =", round(ppi_deg,2)))

## [1] "ppi =" "0.13"

print(c("addhealth =", round(addhealth_deg,2)))

## [1] "addhealth =" "0.07"

print(c("tribes =", round(tribes_deg,2)))

## [1] "tribes =" "0.35"

print("Closeness:")

## [1] "Closeness:"

print(c("ppi =", round(ppi_close,2)))

## [1] "ppi =" "0.26"

print(c("addhealth =", round(addhealth_close,2)))

## [1] "addhealth =" "0.16"

print(c("tribes =", round(tribes_close,2)))

## [1] "tribes =" "0.5"

print("Betweeness:")

## [1] "Betweeness:"

print(c("ppi =", round(ppi_between,2)))

## [1] "ppi =" "0.31"
```

```
print(c("addhealth =", round(addhealth_between,2)))
```

```
## [1] "addhealth =" "0.37"
```

```
print(c("tribes =", round(tribes_between,2)))
```

```
## [1] "tribes =" "0.51"
```

```
print("Eigenvector:")
```

```
## [1] "Eigenvector:"
```

```
print(c("ppi =", round(ppi_ev,2)))
```

```
## [1] "ppi =" "0.33"
```

```
print(c("addhealth =", round(addhealth_ev,2)))
```

```
## [1] "addhealth =" "0.59"
```

```
print(c("tribes =", round(tribes_ev,2)))
```

```
## [1] "tribes =" "0.36"
```

h) For each network, find a measure of the vertex-connectivity and edge-connectivity of the giant component. Find a minimal set of vertices and a minimal set of edges that you could remove to disconnect the giant component.

```
library(igraph)
```

```
## Warning: package 'igraph' was built under R version 3.6.2
```

```
##
## Attaching package: 'igraph'
```

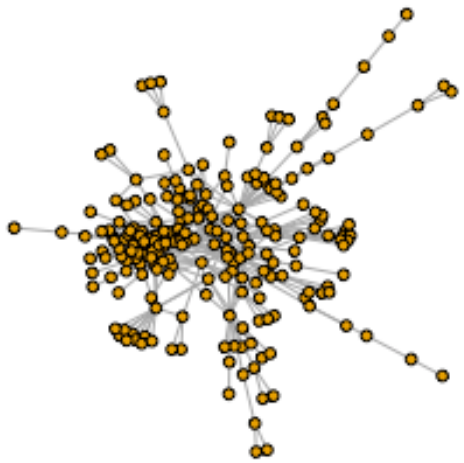
```
## The following objects are masked from 'package:sna':
##
##     betweenness, bonpow, closeness, components, degree, dyad.census,
##     evcent, hierarchy, is.connected, neighborhood, triad.census
```

```
## The following objects are masked from 'package:network':
##
##     %c%, %s%, add.edges, add.vertices, delete.edges, delete.vertices,
##     get.edge.attribute, get.edges, get.vertex.attribute, is.bipartite,
##     is.directed, list.edge.attributes, list.vertex.attributes,
##     set.edge.attribute, set.vertex.attribute
```

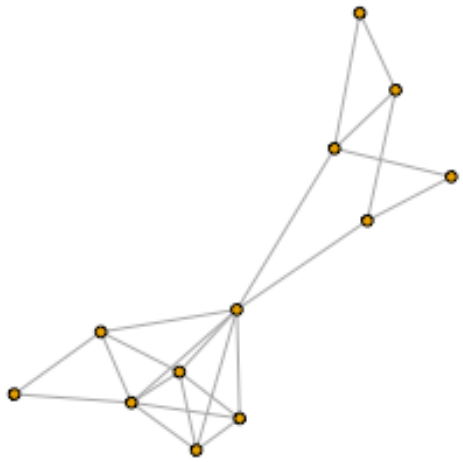
```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

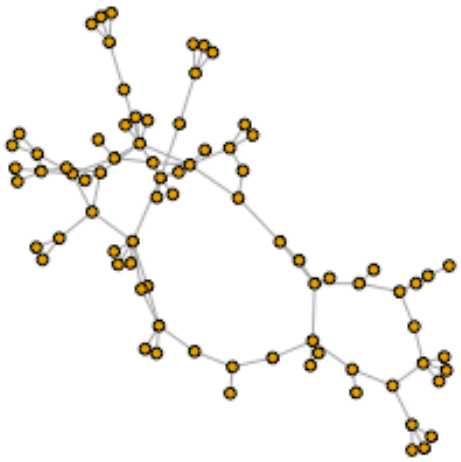
```r
# Function to get largest conncected component
f_CC <- function(g){
  components <- igraph::clusters(g, mode="weak")
  biggest_cluster_id <- which.max(components$csize)
  vert_ids <- V(g)[components$membership == biggest_cluster_id]
  ig = igraph::induced_subgraph(g, vert_ids)
}

# Create igraph objects
ppi_igraph = graph_from_data_frame(butland_ppi,directed = FALSE)
tribes_igraph = as.undirected(graph_from_adjacency_matrix(tribes_pos))
addhealth_igraph = graph_from_data_frame(addhealth9$E[ind,],directed = FALSE)

# Get largest connected components
ppi_igraph_cc = f_CC(ppi_igraph)
tribes_igraph_cc = f_CC(tribes_igraph)
addhealth_igraph_cc = f_CC(addhealth_igraph)

# Plot
plot(ppi_igraph_cc, layout=layout_with_fr, vertex.size=5, vertex.label=NA)
```



```r
plot(tribes_igraph_cc, layout=layout_with_fr, vertex.size=5, vertex.label=NA)
```



```
plot(addhealth_igraph_cc, layout=layout_with_fr, vertex.size=5,
vertex.label=NA)
```



```
# Vertex connectiity.
print("Vertex connectivity:")
```

```
## [1] "Vertex connectivity:"
```

```
vertex_connectivity(ppi_igraph_cc)
```

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

```
vertex_connectivity(tribes_igraph_cc)
```

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

```
vertex_connectivity(addhealth_igraph_cc)
```

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

```
# In each graph, we need only disconnect one node in the giant component in
order to make the graph disconnected.

# Edge connectivity:
print("Edge connectivity:")
```

```
## [1] "Edge connectivity:"
```

```
edge_connectivity(ppi_igraph_cc)
```

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

```
edge_connectivity(tribes_igraph_cc)
```

```
## [1] 2
```

```
edge_connectivity(addhealth_igraph_cc)
```

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

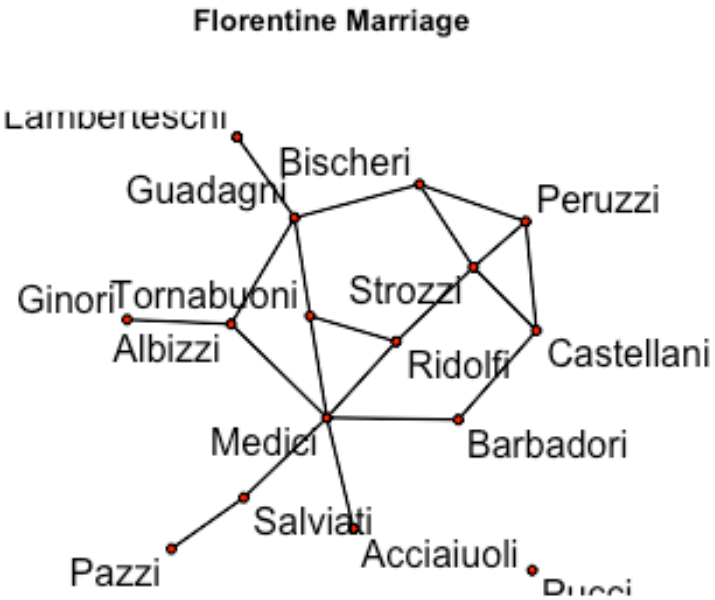
```
# In the ppi and addhealth graphs, we need only disconnect one edge to make
the giant component disconnected. In each of these, there are a number of
tree like projections which we could disconnect. The tribes graph is much
smaller and consists of two clusters connected by two edges. The minimum
number of edges we need to disconnect in this graph is two.
```

2) *Connectivity:* Here we consider the Florentine marriage data from the **network** package, as we did in the class exercise.

```
library(networkdata)
data(florentine)
```

a) Plot the network, with the names labeled.

```
plot(flomarriage,
     main="Florentine Marriage",
     cex.main=0.8,
     label = network.vertex.names(flomarriage))
```



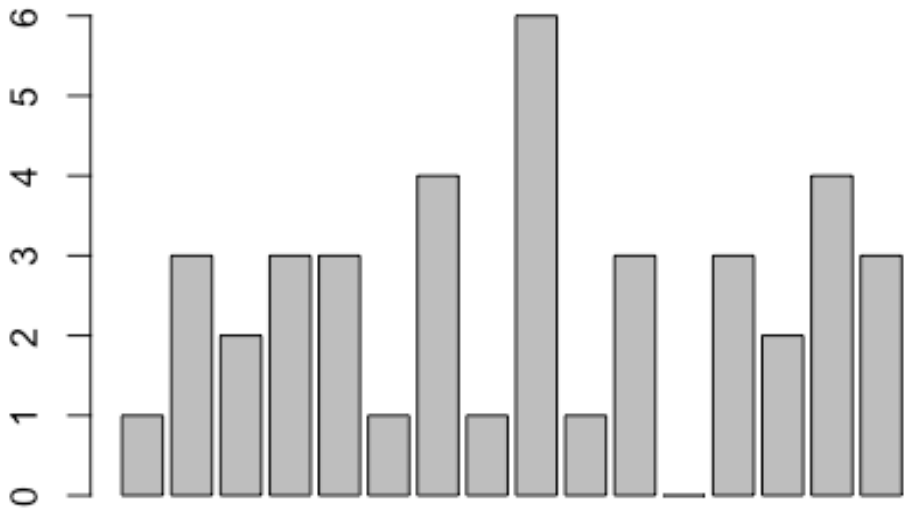
b) Which families have the highest and lowest degrees? What are their degrees?

```
detach("package:igraph", unload=TRUE)
library(sna)
library(network)

# flomarriage_net <- as.network(flomarriage)

# Get family names
flo_names = network.vertex.names(flomarriage)

# Get degrees
flomarriage_deg = degree(flomarriage,gmode="graph")
highest_index = which.max(flomarriage_deg)
lowest_index = which.min(flomarriage_deg)
print(c("The family with the most degrees is", flo_names[highest_index]))

## [1] "The family with the most degrees is" "Medici"

print(c("with a degree of", flomarriage_deg[highest_index]))

## [1] "with a degree of" "6"

print(c("The family with the fewest degrees is", flo_names[lowest_index]))
```

```
## [1] "The family with the fewest degrees is"
## [2] "Pucci"

print(c("with a degree of", flomarriage_deg[lowest_index]))

## [1] "with a degree of" "0"
```

c) Use the **degree()** function in the **sna** package to find the degree distribution. Note that for undirected networks, this function returns twice the degrees. You may also find the **table** command helpful.

```
flomarriage_deg = degree(flomarriage,gmode="graph")

# Plot bar graph of degrees
barplot(flomarriage_deg)
```




```
# Graph with node sizes scaled by degrees
gplot(flomarriage , vertex.cex=(flomarriage_deg )/2, vertex.sides=50,label =
flo_names,gmode = "graph")
```

d) Compute the transitivity of the network by hand. You may check yourself using the **gtrans()** command in **sna**.

```
# There are 3 loops of length 3, so there are 6*3 = 18 paths of length 3.
# There are 94 paths of length 2: for 5 Acciaiuoli, 3 for Albizzi, 7 for
Barbadori
# 8 for Bischeri, 6 for Castellani, 3 for Ginori, 6 for Guadagni, 2 for
Lamberteschi,
# 9 for Medici, 1 for Pazzi, 7 for Peruzzi, 0 Pucci, 8 for Ridolfi, 5 for
Salviati,
# 10 for Strozzi, 8 for Tornabuoni.

print("The transitivity is:")

## [1] "The transitivity is:"

transitivity = 18/94
print(transitivity)

## [1] 0.1914894

gtrans(flomarriage)

## [1] 0.1914894
```

e) Which family is excluded from the large component? Pucci is excluded from the large component.

f) Find measures of the vertex-connectivity and edge-connectivity of the large component. Find a minimal set of vertices and a minimal set of edges that you could remove to disconnect the giant component.

```
library(igraph)
```

```
## Warning: package 'igraph' was built under R version 3.6.2
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:sna':
##
##     betweenness, bonpow, closeness, components, degree, dyad.census,
##     evcent, hierarchy, is.connected, neighborhood, triad.census
```

```
## The following objects are masked from 'package:network':
##
##     %c%, %s%, add.edges, add.vertices, delete.edges, delete.vertices,
##     get.edge.attribute, get.edges, get.vertex.attribute, is.bipartite,
##     is.directed, list.edge.attributes, list.vertex.attributes,
##     set.edge.attribute, set.vertex.attribute
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```
# Create igraph object
x = as.edgelist(flomarriage, attrname = NULL, as.sna.edgelist = FALSE)
el = x[,1:2]
flor_igraph = graph_from_edgelist(el, directed = FALSE)

# Function to get largest conncected component
f_CC <- function(g){
  components <- igraph::clusters(g, mode="weak")
  biggest_cluster_id <- which.max(components$csize)
  vert_ids <- V(g)[components$membership == biggest_cluster_id]
  ig = igraph::induced_subgraph(g, vert_ids)
}

# Remove Pucci family to get the largest connected component:
flor_cc = f_CC(flor_igraph)
```

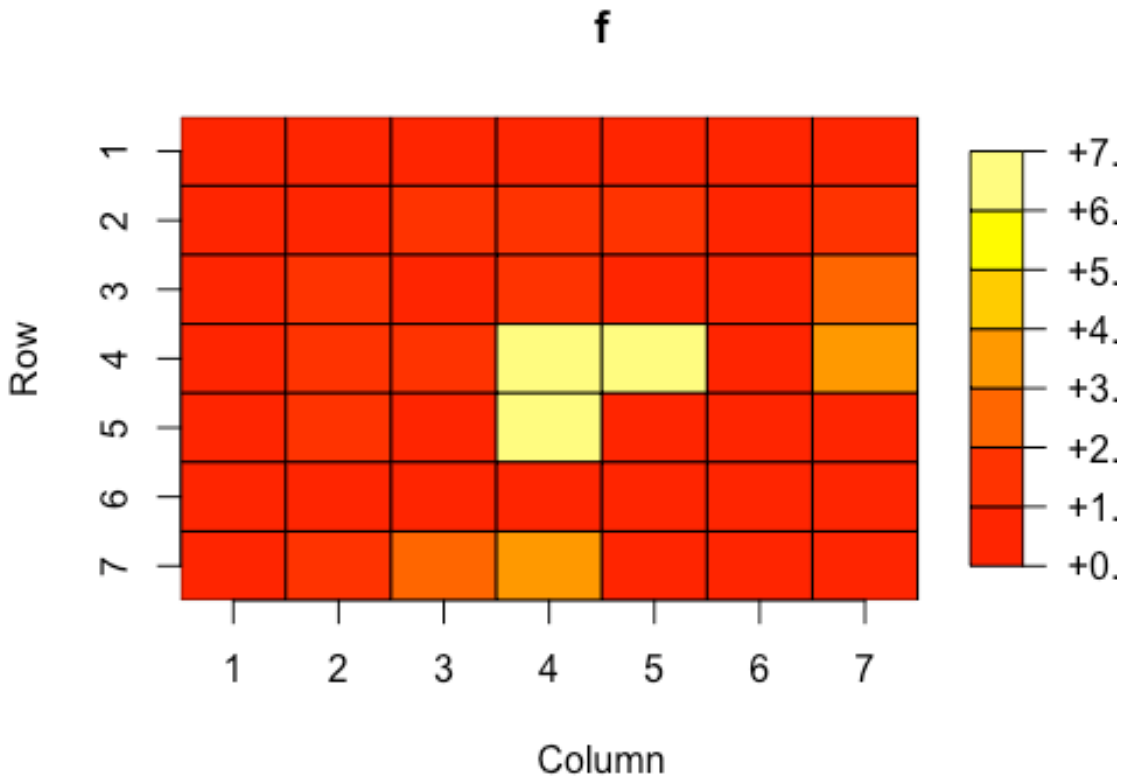
```r
# Plot the largest connected component:
plot(flor_cc,layout=layout_with_fr,vertex.size=5,vertex.label=NA)
```

```r
# Vertex connectivity.
print("Vertex connectivity:")

## [1] "Vertex connectivity:"

vertex_connectivity(flor_cc)

## [1] 1

# Edge connectivity:
print("Edge connectivity:")

## [1] "Edge connectivity:"

edge_connectivity(flor_cc)

## [1] 1
```

g) Find the joint degree distribution and the degree correlation (there is no **sna** function to do this, and you may need to look up the definition of these terms).

```r
# This function computes the joint degree distribution of g
f_Distribution <- function(g){
```

```
  adj = as.matrix(as_adjacency_matrix(g)) # Adjacency mat
  deg_v = colSums(adj) # Degrees
  max_deg = max(deg_v)+1 # Max degree
  f = matrix(0,max_deg,max_deg) # Initialize distribution

  # Compute distribution
  for (i in 1:dim(adj)[1]){
    for (j in 1:dim(adj)[2]){
      di = deg_v[i]+1
      dj = deg_v[j]+1
      if (adj[i,j] == 1){
        f[di,dj] = f[di,dj] + 1
      }
    }
  }
  f = (f + t(f))/2
}

f = f_Distribution(flor_igraph)

library(plot.matrix)

## Warning: package 'plot.matrix' was built under R version 3.6.2

plot(f)
```



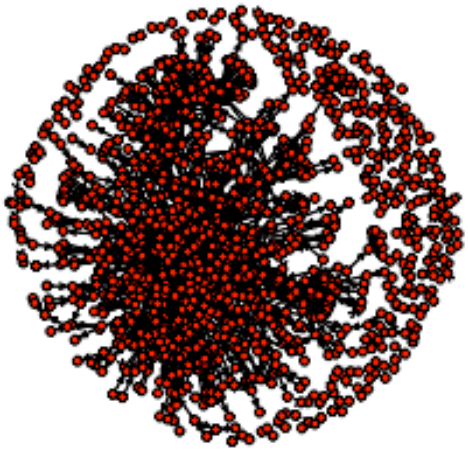

```
# Computes degree correlation:
f_corr <- function(g) {
        el = get.edgelist(g)
        d1 = degree(g,el[,1])
        d2 = degree(g,el[,2])
        if(sd(c(d1,d2)) == 0){return(1)}
        corr = cor(d1,d2,method="pearson")
        return(corr)}

print("The degree correlation is:")

## [1] "The degree correlation is:"

f_corr(flor_igraph)

## [1] -0.3716231
```

3) *Degree distributions:* Degree distributions summarize the densities of ties of the population of nodes. In this question we explore the interactions between proteins of the yeast *S. cerevisiae*. The nodes are types of proteins in the yeast and a directed tie is said to exist if a protein binds to the target protein in a iwet lab" experiment set up to test just this. Not all protein combinations are tested. Here we will

consider a series ofmapping" experiments conducted in 2008 that covered approximately 20% of all yeast binary interactions ()

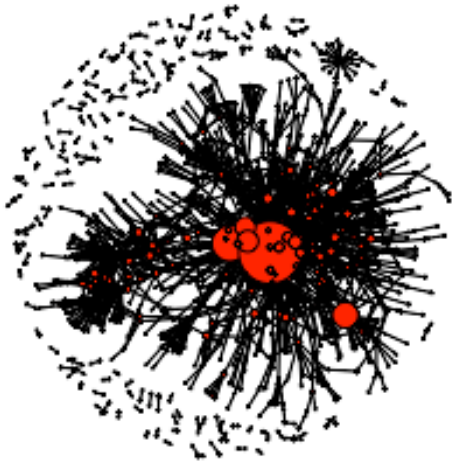
a) Go to the home page of the ``Yeast Interactome Project'': http://interactome.dfci.harvard.edu/S_cerevisiae/ From there download the interactions from CCSB-YI11. These comprise 1809 interactions among 1278 proteins. Construct a **network** object from this edge-list.

```
YI11 <- read.delim("/Users/peterracioppo/Desktop/CCSB-Y2H.txt")

f_Network <- function(data,dir,mat_type){
  net <- as.network(x = data, # the network object
                    directed = dir, # specify whether the network is directed
                    loops = FALSE, # do we allow self ties (should not allow
them)
                    matrix.type = mat_type # the type of input
                       )
}

# Directed graph
YI11_net = f_Network(YI11,TRUE,"edgelist")

# Plot directed graph
plot(YI11_net, vertex.cex = 1)
```

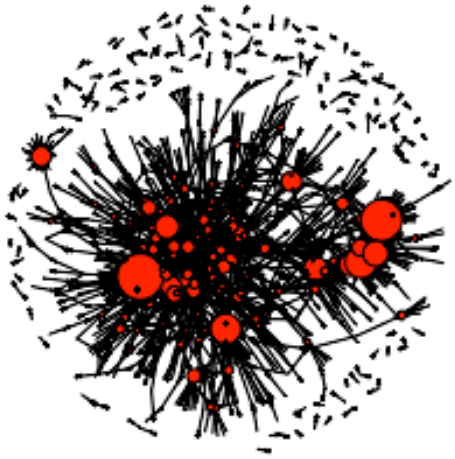


b) Construct the out-degree sequence for the network. Construct the in-degree sequence. Are the in-and out-degrees correlated? Use the sum of the in-degree and out-degree as an overall measure of the proteins activity (which we will refer to as its degree).

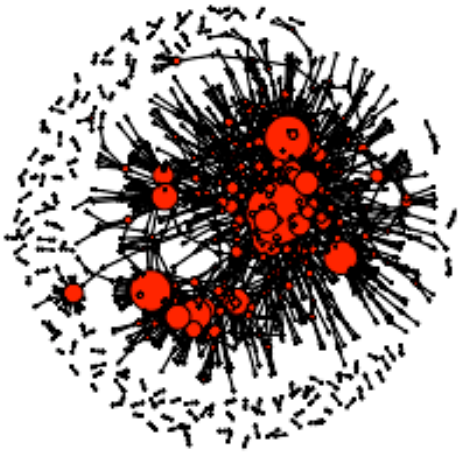
```
detach("package:igraph", unload=TRUE)
library(sna)
library(network)

YI11_out = degree(YI11_net,gmode="digraph",cmode="outdegree") # Get out
degree
YI11_in = degree(YI11_net,gmode="digraph",cmode="indegree") # Get in degree
YI11_deg = YI11_out + YI11_in

deg_cor = cor(YI11_out,YI11_in)
print(c("Correlation =", deg_cor))

## [1] "Correlation ="       "-0.0685780170639995"

# Graphs with node sizes scaled by degrees
gplot(YI11_net, vertex.cex=(YI11_out)^0.8/2, vertex.sides=50)
```



```
gplot(YI11_net, vertex.cex=(YI11_in)^0.8/2, vertex.sides=50)
```



```
gplot(YI11_net, vertex.cex=(YI11_deg)^0.8/2, vertex.sides=50)
```

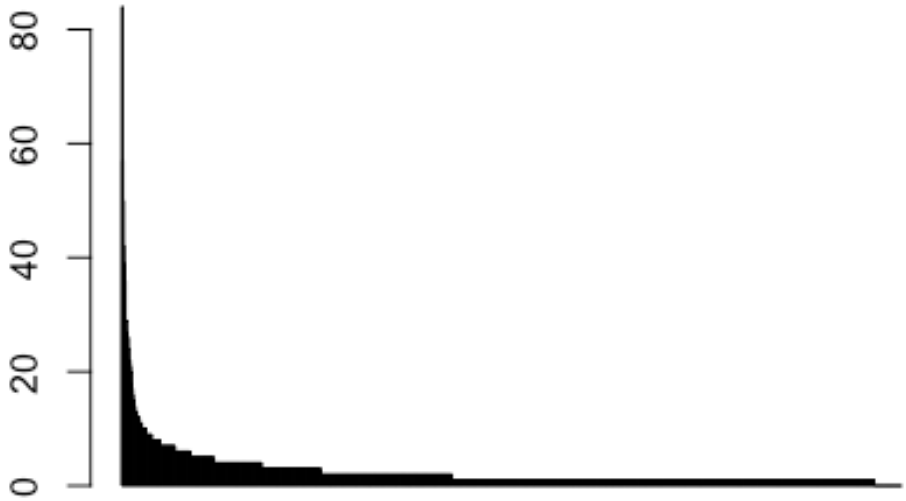


c) Fit degree distribution models using the **degreenet** package. Fit the classical discrete Pareto/Zipf law model discussed in the books:

$$P(K = k; \nu) = \frac{k^{-\nu}}{\zeta(\nu)} \qquad \nu \geq 1, k = 1,2,\ldots$$

d) Fit the Yule, Waring, Poisson, and Conway-Maxwell-Poisson models. Note: The corresponding functions are **ayulemle**, **adpmle**, **ayulemle**, **awarmle**, **apoimle**, **acmpmle**. See the help pages.

```
library(degreenet)

## degreenet: Models for Skewed Count Distributions Relevant to Networks
## Version 1.3-3 created on 2018-03-25.
## copyright (c) 2013, Mark S. Handcock, University of California - Los
Angeles
##  Based on "statnet" project software (statnet.org).
##  For license and citation information see statnet.org/attribution
##  For citation information, type citation("degreenet").
##  Type help("degreenet-package") to get started.

# Plot bar graphs of degrees
barplot(rev(sort(YI11_deg)))
```




```
# Fit models
pareto = adpmle(YI11_deg)
yule = ayulemle(YI11_deg)
waring = awarmle(YI11_deg)
poisson = apoimle(YI11_deg)

## Warning in stats::optim(par = guess, fn = llpoi, hessian = hessian,
control = list(fnscale = -10), : one-dimensional optimization by Nelder-Mead
is unreliable:
## use "Brent" or optimize() directly

cmp = acmpmle(YI11_deg)
```

d) How can we compare the fits of the different models? Use the **lldpall()**,etc, functions to compute the corrected AIC and the BIC for the models. Summarize the fits of the models in a table. Which models fit best? Briefly comment on the models as a whole in terms of their fit.

```
# Compute AIC/BICs
pareto_ll = lldpall(pareto$theta,YI11_deg)
yule_ll = lldpall(yule$theta,YI11_deg)
waring_ll = lldpall(waring$theta[1],YI11_deg)
poisson_ll = lldpall(poisson$theta,YI11_deg)
cmp_ll = lldpall(cmp$theta[1],YI11_deg)
```

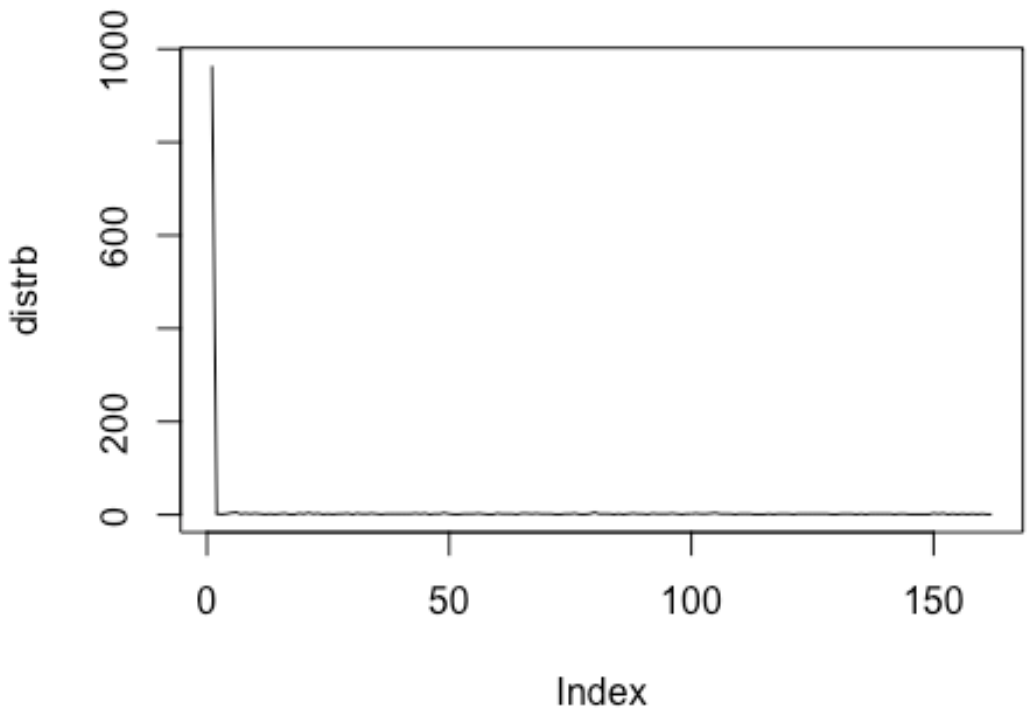
```
# Get AIC/BICs
pareto_AIC = pareto_ll[3]
pareto_BIC = pareto_ll[4]
yule_AIC = yule_ll[3]
yule_BIC = yule_ll[4]
waring_AIC = waring_ll[3]
waring_BIC = waring_ll[4]
poisson_AIC = poisson_ll[3]
poisson_BIC = poisson_ll[4]
cmp_AIC = cmp_ll[3]
cmp_BIC = cmp_ll[4]

# Create table
AIC = c(pareto_AIC,yule_AIC,waring_AIC,poisson_AIC,cmp_AIC)
BIC = c(pareto_BIC,yule_BIC,waring_BIC,poisson_BIC,cmp_BIC)
table = rbind(AIC,BIC)
colnames(table) <- c("Pareto","Yule","Waring","Poisson","CMP")
print(table)

##      Pareto     Yule   Waring  Poisson      CMP
## AIC 4335.99 4324.513 4404.387 4490.614 4490.585
## BIC 4351.43 4339.954 4419.827 4506.054 4506.025

# Which model has minimum AIC/BIC
which.min(AIC)

## AICC
##    2

which.min(BIC)

## BIC
##   2

# The Yule model has the lowest AIC and BIC, which indicates that it's the
best fit. The
# Pareto model is only slightly worse.
# The Waring model has the third lowest AIC and BIC, while the Poisson and
# Conway-Maxwell-Poisson models have AIC and BIC values which are almost
exactly equal to each other, and higher than the other two.
```

3) *Components:* Continuing with the CCSB-YI11 network: Consider undirected the version of it where two proteins are linked if either of them binds with the other. This network has 1809 edges.

a) Compute the component distribution of the network. How large is the largest component? Does the network have a ``giant" component? Note: Consider the **component.dist** function in the **sna** package.

```
# Undirected graph
YI11_un = f_Network(YI11,FALSE,"edgelist")
```

```
comp_dist = component.dist(YI11_un,connected="strong")
distrb = comp_dist$csize
membership = comp_dist$membership

# Plot the distribution:
plot(distrb,type="l")
```




```
print(c("The largest connected component has", max(distrb),"vertices."))
```

```
## [1] "The largest connected component has" "964"
## [3] "vertices."
```

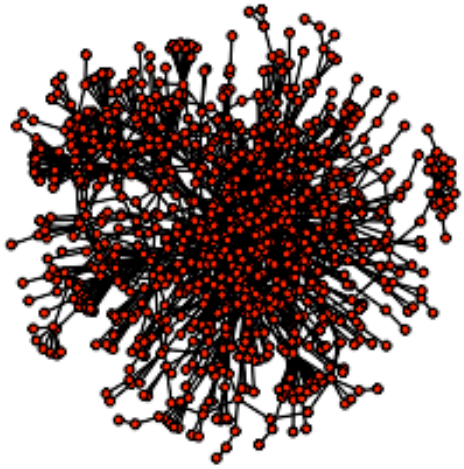
```
print(c("The second largest connected component has", max(distrb[-
1]),"vertices."))
```

```
## [1] "The second largest connected component has"
## [2] "5"
## [3] "vertices."
```

```
# The network has a giant component. The giant component has 964 vertices,
# whereas the second largest component has 5 vertices.
```

b) Plot the subgraph comprised of the largest component.

```
largest_comp = component.largest(YI11_un,connected="strong",result="graph")
large_comp_net = f_Network(largest_comp,FALSE,"adjacency")
plot(large_comp_net)
```



c) Compute the (pairwise) matrix of geodesic distances between the proteins. Create a summary tabulation of the distances. What proportion of nodes-pairs are reachable (from each other)? What is the mean geodesic distance for reachable pairs? How many isolates are there in the network?

```
# This function computes the geodesic distance of a graph
f_Geodesic <- function(m){
  # Input: adjacency matrix m
  # Output: matrix of pair-wise geodesic distances
  x = m
  d = m

  for (i in 2:(dim(m)[1]-1)){
    x = x%*%m
    d = d + i*(x>0) - ((d + i*(x>0))>i)*(i*(x>0))
    if (min(m) > 0){break}
  }
  return(d)
}
```

```r
M1 <- matrix(rpois(36,1),nrow=6)>0
f_Geodesic(M1)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    2    2    1    2    1
## [2,]    2    1    1    1    2    2
## [3,]    1    1    1    2    1    1
## [4,]    1    1    1    2    1    2
## [5,]    1    2    1    2    1    2
## [6,]    1    2    1    1    1    2
```

```r
M = as.matrix.network.adjacency(YI11_un)
# protein_geodesics = f_Geodesic(M)

# My function works, but it's too slow for this massive graph.

library(igraph)
```

```
## Warning: package 'igraph' was built under R version 3.6.2
```

```
##
## Attaching package: 'igraph'
```

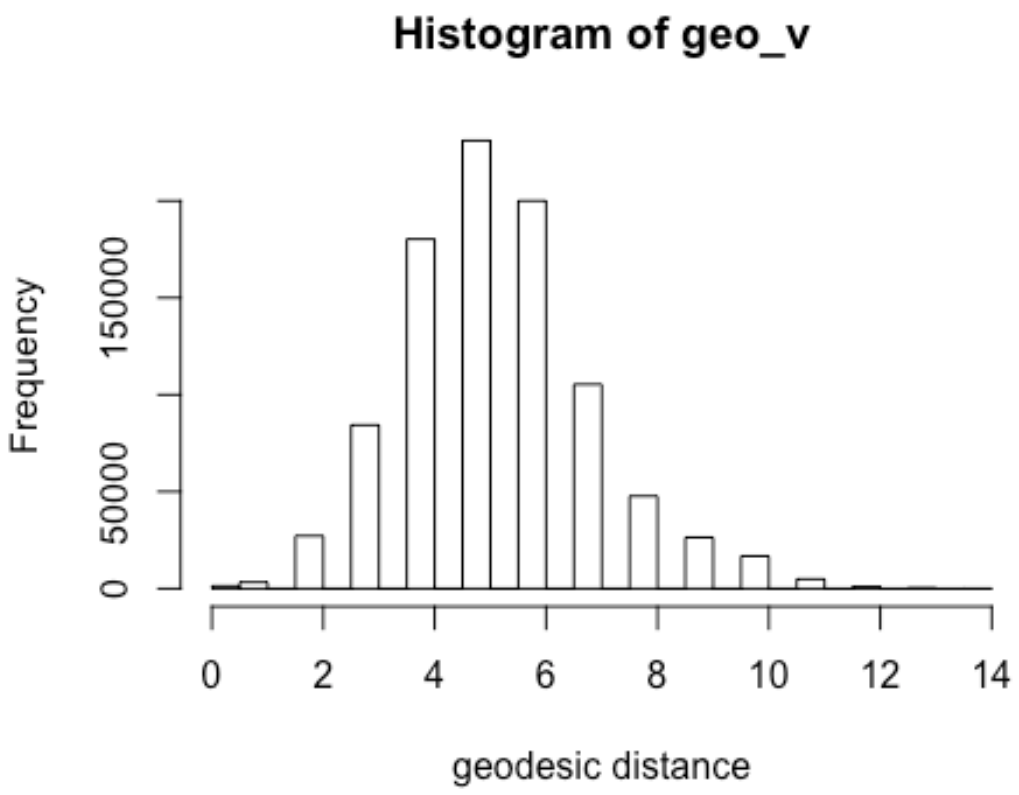
```
## The following objects are masked from 'package:sna':
##
##     betweenness, bonpow, closeness, components, degree, dyad.census,
##     evcent, hierarchy, is.connected, neighborhood, triad.census
```

```
## The following objects are masked from 'package:network':
##
##     %c%, %s%, add.edges, add.vertices, delete.edges, delete.vertices,
##     get.edge.attribute, get.edges, get.vertex.attribute, is.bipartite,
##     is.directed, list.edge.attributes, list.vertex.attributes,
##     set.edge.attribute, set.vertex.attribute
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```r
# Create igraph object
YI11_igraph = graph_from_data_frame(YI11,directed = FALSE)

geo_dist = distances(YI11_igraph)
geo_v = as.vector(geo_dist)

# Histogram of geodesic distances:
hist(geo_v,xlab="geodesic distance")
```




```
N_t = (dim(geo_dist)[1]**2 - dim(geo_dist)[1])/2 # Number of vertex pairs,
excluding self-loops
unreachable = sum(geo_dist == Inf)/2 # Number of pairs of vertices which are
unreachable from each other
reachable = N_t - unreachable # Number of pairs which are reachable from each
other
prop_reach = reachable/N_t # Proportion of reachable pairs
print(c("The proportion of nodes-pairs which are reachable from each other
is",round(prop_reach,4)))
```

```
## [1] "The proportion of nodes-pairs which are reachable from each other is"
## [2] "0.5691"
```

```
reachable_geo = geo_v[geo_v != Inf] # Set of reachable pairs of nodes
reachable_geo = reachable_geo[reachable_geo != 0] # Remove self loops
mean_geo = mean(reachable_geo) # Mean geodesic distance among reachable nodes
print(c("The mean geodesic distance for the reachable pairs
is",round(mean_geo,4)))
```

```
## [1] "The mean geodesic distance for the reachable pairs is"
## [2] "5.3651"
```

```
isoM = (geo_dist == Inf) # Indicator matrix of unreachable nodes
L = dim(geo_dist)[1]-1 # Matrix dimension - 1
```

```r
N_isolates = sum((colSums(isoM) == L)) # Column sum of indicator matrix gives the total number of isolates
print(c("There are",N_isolates,"isolates."))
```

```
## [1] "There are" "45"        "isolates."
```

Submit

Upload the PDF version of your homework before the end of the due date via the *Homework* page on the class CCLE using **Gradescope**. Submit only the PDF file, but make sure RStudio knits it correctly from your Rmd file. The R code must be clearly readable and properly commented. Explain your solutions well.