National Research University Higher School of Economics, Faculty of Social Sciences

Introduction to Network Analysis, Spring 2019

Seminar 6 and Homework 4 Assignment, 22/02/2019

Dmitry Zaytsev, PhD and Valentina Kuskova

(with deep appreciation to all used sources; references available in text and upon request)

Welcome to the sixth seminar! Hard to believe it, but we are half-way through the course already. Before attempting to run any code, please make sure you have every necessary package installed; however, pay attention. Today we will learn that some packages interfere with each other, so you will need to unload some before you can use others.

Contents of today's seminar:

- In your Seminar 6 folder, you will find the following files:
  1. This file, "Seminar 6 and Homework 4," in .pdf format.
  2. Data files "formal.csv" and "roles.csv."
- For today's assignment, you will need the following packages (remember, R is case sensitive, make sure, when installing packages, to keep the appropriate case:
  1. rmarkdown
  2. RColorBrewer
  3. NetData
  4. network
  5. sna

  Remember that to use a certain package, you need to make sure that the library is installed (using install.packages("packagename") command). To use a loaded package, you call it with "library.(packagename)" command.
- After completing today's seminar, you should be able to accomplish the following:
  1. Continue to enhance your R programming skills: learn simple loops.
  2. Learn how to perform blockmodeling analysis.
  3. Acquire additional skills for network graphs.
  4. Compare and interpret obtained network results. That's right, you are already network analysts!

Seminar 6 assignment. Please reproduce the R code in this document and answer the questions that are marked as *Assignment questions*. Please include your answers right after each question in your RMarkdown file. You should submit both the .Rmd and the .pdf files with results.

Homework 4 assignment is provided at the end of this document.

Both assignments are due on Friday, March 1, 23.59.

Experimenting with R-code: loops

Please make sure your working directory is set to a folder where the data files are located; otherwise, you will be forced to provide a full path to data every time.

Extracting networks from data frames

For this assignment, you will need the *NetData* package, so install the package (remember - do it in Console!) and call it from RMarkdown. This package contains a number of datasets, which can be used for labs and practice - if you are looking for more data to practice on your own, check this package out (using the "?NetData" command in Console). We are using the kracknets dataset, which, actually, is the same as the HiTech dataset I've given you already.

```
##install.packages("NetData")
library(NetData)

#Pull the dataset out:
data(kracknets, package = "NetData")
```

In your Global Environment, you now see three sets of data, all in format "name"_data_frame. So, we have three different datasets in one, and they are obviously dataframes. Let's look at what these datasets are:

```
head(advice_data_frame)
```

```
##   ego alter advice_tie
## 1   1     1          0
## 2   1     2          1
## 3   1     3          0
## 4   1     4          1
## 5   1     5          0
## 6   1     6          0
```

```
head(friendship_data_frame)
```

```
##   ego alter friendship_tie
## 1   1     1              0
## 2   1     2              1
## 3   1     3              0
## 4   1     4              1
## 5   1     5              0
## 6   1     6              0
```

```
head(reports_to_data_frame)
```

```
##   ego alter reports_to_tie
## 1   1     1             0
## 2   1     2             1
## 3   1     3             0
## 4   1     4             0
## 5   1     5             0
## 6   1     6             0
```

So obviously, we have ego networks of people who give advice to each other, who are friends, and who reports to whom. From these three dataframes, we can make an array of data, combining all three dataframes into one.

```
# First, make an array of 3 the 3 data frames
krack <- list(advice_data_frame,
              friendship_data_frame,
               reports_to_data_frame)

## add names to objects in list
graphs <- c('advice','friendship','reports')
names(krack) <- graphs
## check on list
length(krack) #how many elements we have in our krack dataset
```

```
## [1] 3
```

```
names(krack)
```

```
## [1] "advice"     "friendship" "reports"
```

"For" loops

Now we need to turn each element into a matrix. We already know how to extract elements from a matrix, and we can certainly do it here one element at a time. We have only three - it won't take too much time. But what if we have a dozen? Two dozen? It's the same code over and over, only one element (matrix number) will change. In programming, repetitive tasks that are almost identical are executed with the help of a *loop* - and there are several types. Below is a simple *"for" loop*, which contains several elements: 1. for (i in n) - how long to repeat the loop and for what elements. Here, we tell R that code below needs to be executed for the ith element in n total elements. 1. {code to execute}

R recognizes the "for" loop and will increment i after completing the code - it will start with number 1 (first i), run the code, increment the i to the second i, run the code, and so on until it's done (reaches the end, n). In our case, the loop will only run three times:

```
for (i in 1:length(krack)){
  krack[[i]] <- as.matrix(krack[[i]])
}
```

Loops are generally very useful, so we encourage you to familiarize yourselves with them.

Another useful command is a "subset" command - it can take only a part of data. For analyzing network structure, once we've examined the network composition, it is sometimes helpful to remove nodes with zero edges - meaning, nodes that are isolates, not connected to anyone. This is what we do with the loop below (notice the syntax of the *subset* command:

```
for(i in 1:3){
krack[[i]] <- subset(krack[[i]],
                (krack[[i]][,3] > 0 ))
}
dim(krack[[1]]) # What is the size now?
```

```
## [1] 190   3
```

```
# Quick look:
head(krack[[1]])
```

```
##      ego alter advice_tie
## [1,]   1     2          1
## [2,]   1     4          1
## [3,]   1     8          1
```

```
## [4,]   1    16          1
## [5,]   1    18          1
## [6,]   1    21          1
```

Now, we make three graphs of our networks, and you should already understand what we do with each of the commands below:

```
names(attributes)
```

```
## [1] "AGE"    "TENURE" "LEVEL"  "DEPT"
```

```
library(network)
```

```
## network: Classes for Relational Data
## Version 1.13.0.1 created on 2015-08-31.
## 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.
```

```
for (i in 1:3){
krack[[i]] <- network(krack[[i]],
                matrix.type = 'edgelist',
                  vertex.attr = list(attributes[,1], attributes[,2],
                                     attributes[,3], attributes[,4]),
                  vertex.attrnames = list("AGE","TENURE","LEVEL","DEPT"))
}
advice <- krack$advice
friendship <- krack$friendship
reports <- krack$reports


# Check networks
advice
```

```
##  Network attributes:
##   vertices = 21
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 190
##     missing edges= 0
##     non-missing edges= 190
##
##  Vertex attribute names:
##     AGE DEPT LEVEL TENURE vertex.names
##
## No edge attributes
```

```
friendship
```

```
##  Network attributes:
##   vertices = 21
```

```
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 102
##     missing edges= 0
##     non-missing edges= 102
##
##  Vertex attribute names:
##     AGE DEPT LEVEL TENURE vertex.names
##
## No edge attributes
```

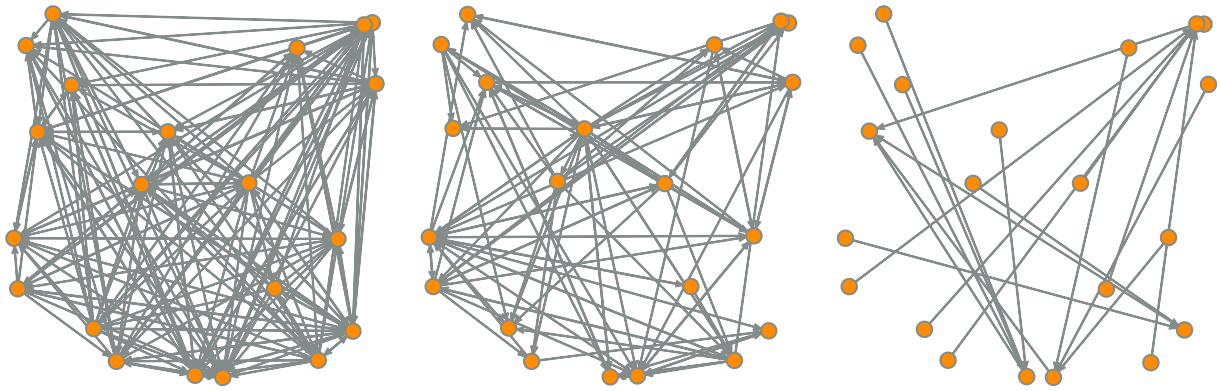
```
reports
```

```
##  Network attributes:
##   vertices = 21
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 20
##     missing edges= 0
##     non-missing edges= 20
##
##  Vertex attribute names:
##     AGE DEPT LEVEL TENURE vertex.names
##
## No edge attributes
```

```
# Let's take a look at these data.
# First, we create a set of coordinates to run the plot in.
# Detailed explanation of what we do here is provided in the answers to the third seminar.
n<-network.size(advice)
v1<-sample((0:(n-1))/n) #create a vector of random numbers
v2<-sample(v1)
x <- n/(2 * pi) * sin(2 * pi * v1)
y <- n/(2 * pi) * cos(2 * pi * v2)
mycoord <- cbind(x,y)

# Plot networks:
par(mar=c(0,0,1,0))
par(mfrow=c(1,3))
plot(advice, edge.col='azure4', vertex.col='darkorange',
     vertex.border='azure4',vertex.cex=2,coord=mycoord,
     main ='Advice')
plot(friendship, edge.col='azure4', vertex.col='darkorange',
     vertex.border='azure4',vertex.cex=2, coord=mycoord,
     main ='Friendship')
plot(reports, edge.col='azure4', vertex.col='darkorange',
     vertex.border='azure4',vertex.cex=2, coord=mycoord,
     main='Direct Reports')
```

Advice **Friendship** **Direct Reports**



Assignment task. For the networks we've obtained, please calculate the following:

1. Dyad census
2. Different kinds of reciprocity
3. Triad census
4. Transitivity
5. Paths
6. Cycles
7. Cliques

Having performed the calculations, please compare your results for each network and make appropriate inferences.

Blockmodeling

Now, we started experimenting with building models out of our data. Remember that blockmodeling is rearrangement of data based on some *a priori* theoretical attribute, such as role or position in the network. We are going to look at building blockmodels from two datasets, "formal" - an adjacency matrix and "roles" - an attribute table where roles for each member of the "formal" group have been set in advance. In "real life" and with real data, you will be selecting roles yourself, and this is called the "exploratory" blockmodeling - we will learn how to do that, too.

A priori blockmodel

For this part of our work, we will need packages *network* and *sna*. Read the data:

```
library(network)
library(sna)
```

```
## Loading required package: statnet.common
```

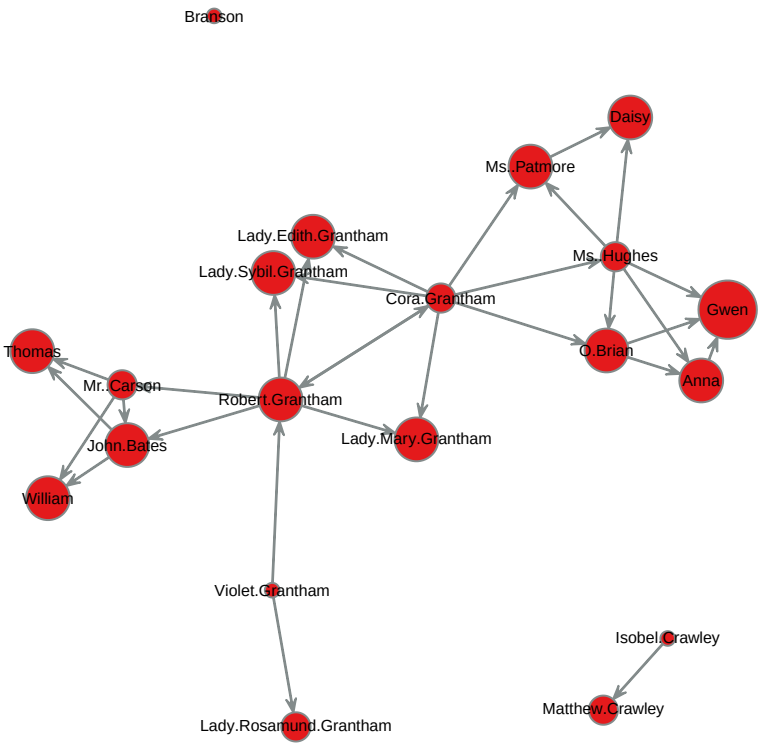
```
##
## Attaching package: 'statnet.common'
```

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

```
## sna: Tools for Social Network Analysis
## Version 2.4 created on 2016-07-23.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##  For citation information, type citation("sna").
##  Type help(package="sna") to get started.
```

```
formal<-as.matrix(read.csv("formal.csv", header = TRUE, row.names=1))
roles<-read.csv("roles.csv", header=TRUE, row.names=1)

formalnet <- network(formal)
par(mar=c(0,0,2,0))
indeg <- degree(formalnet, cmode = 'indegree')
mycoord <- plot(formalnet, displaylabels=TRUE, edge.col='azure4',
                vertex.col="#E41A1C", vertex.border='azure4',
                vertex.cex = indeg + 1 , main ='Downton Abbey',
                label.cex=0.5, label.pos = 5)
```

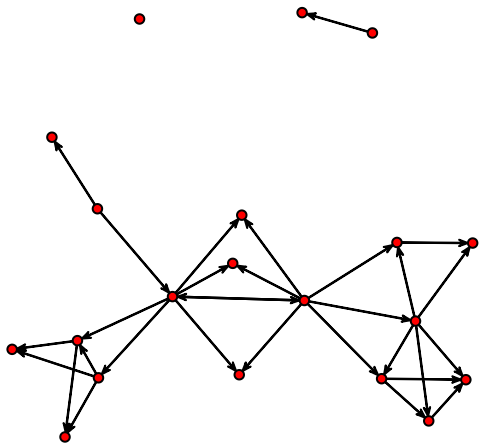


Next, we need to symmetrize our data. Remember how we symmetrized the matrix before? We can symmetrize the graph the same way. Also remember that symmetrizing the network is the transformation of a directed/asymmetric one-mode network into an undirected/symmetric one-mode network. There are also two symmetry types:

1. Strong: $i <->j$ iff (means "if and only if") $i->j$ and $j->i$ (we also refer to it as the "AND rule").
2. Weak: $i <->j$ iff $i->j$ or $j->i$ (the "OR rule").

We also know the drawbacks and benefits of symmetrizing, and one of the benefits, among others, is the ability to build meaningful blockmodels. If we look at the network without the bells and whistles I've added to the plot above, this is what it looks like:

```
plot(formalnet)
```



Obviously, it's a directed network with both one-directional and mutual ties, so we can use both the AND and the OR rules for symmetrizing the network.

```
orRule <- symmetrize(formalnet, rule='weak')   # "or" rule
class(orRule) # symmetrize transformed the network into a matrix
```

```
## [1] "matrix"
```

```
orRule <- network(symmetrize(formalnet, rule='weak'),
                  directed = FALSE) # 'or' rule
class(orRule) # network
```

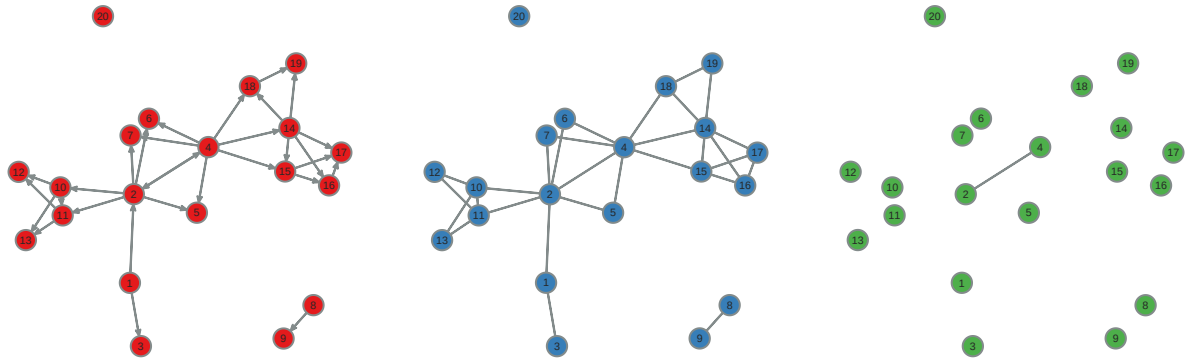
```
## [1] "network"
```

```
andRule <- network(symmetrize(formalnet, rule='strong'),
                   directed = FALSE) # 'and' rule
```

Let's look at what we have as a result:

```
par(mar=c(1,1,2,1))
par(mfrow=c(1,3))
plot(formalnet, main = 'Original', coord=mycoord, vertex.cex =3,
     edge.col='azure4', vertex.col="#E41A1C", vertex.border='azure4',
     label=seq(1:20),label.pos=5,label.cex=.5,label.col='gray15')
plot(orRule, main = 'Or Rule', coord=mycoord, vertex.cex =3,
     edge.col='azure4', vertex.col="#377EB8", vertex.border='azure4',
     label=seq(1:20),label.pos=5,label.cex=.5,label.col='gray15')
plot(andRule, main = 'And Rule', coord=mycoord, vertex.cex =3,
     edge.col='azure4', vertex.col="#4DAF4A", vertex.border='azure4',
```

```
     label=seq(1:20),label.pos=5,label.cex=.5,label.col='gray15')
```

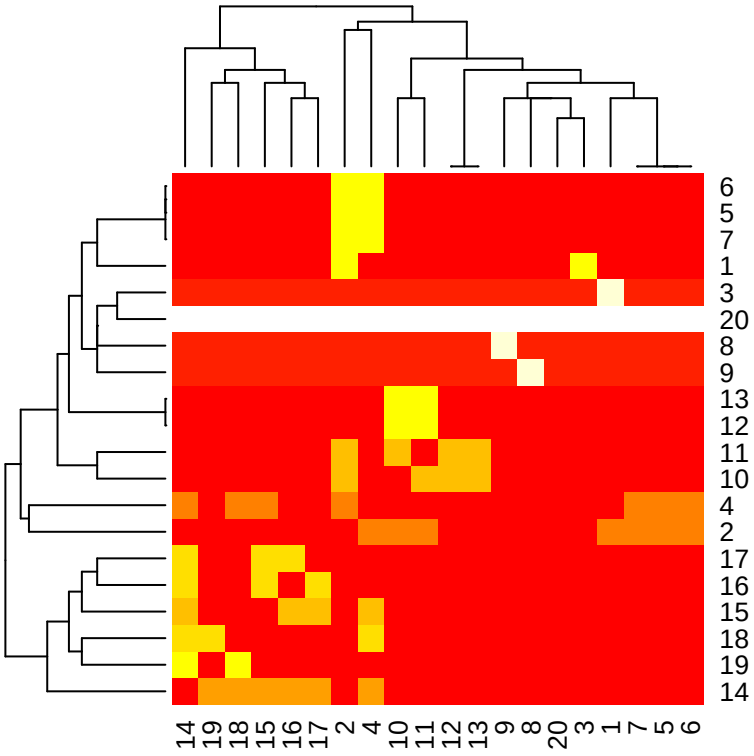


Now, let's create our first blockmodel based on the community detection output (in our file "roles," it's the last column, called "commdetect"). We will call this "a priori" formal blockmodel, because we are building a block models on roles that have been predetermined.

```
# A more descriptive name, so we don't get confused:
snasymmformal <- orRule

aprioriformal<-blockmodel(snasymmformal, roles$commdetect,
                          block.content="density", mode="graph",
                          diag=FALSE)

# We can build what is called a heatmap, showing the relationships between blocks in color:
heatmap(aprioriformal[[4]])
```



Now, that may have looked cool, but what does that all mean? We can color nodes by blocks they belong to.

```
# Let's visualize the network with nodes colored by block.
# These are our blocks.
aprioriformal[[1]]
```

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

```
aprioriformal[[2]]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
aprioriformal[[3]]
```

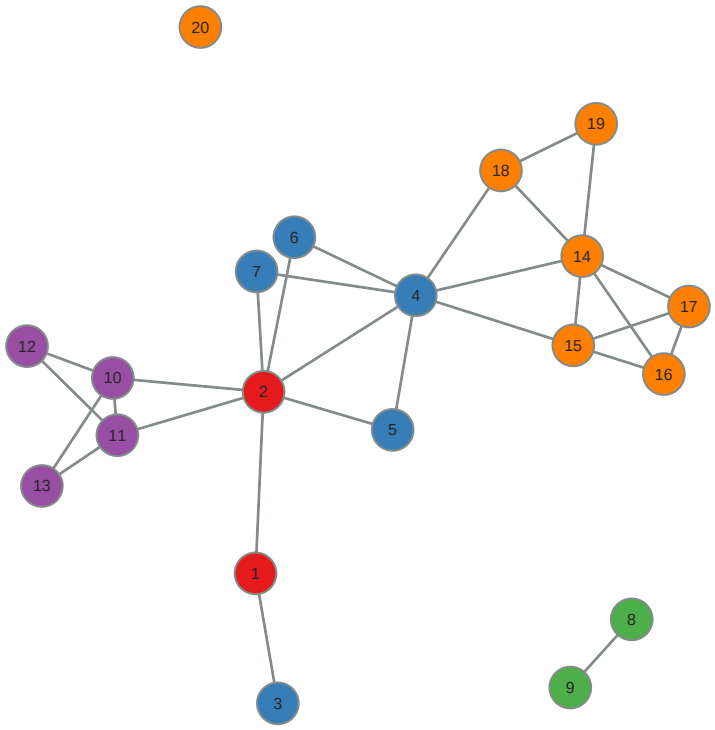
```
## [1] "density"
```

```
aprioriformal[[4]]
```

```
##    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
## 1  0 1 1 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 2  1 0 0 1 1 1 1 0 0  1  1  0  0  0  0  0  0  0  0  0
## 3  1 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 4  0 1 0 0 1 1 1 0 0  0  0  0  0  1  1  0  0  1  0  0
## 5  0 1 0 1 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 6  0 1 0 1 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 7  0 1 0 1 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 8  0 0 0 0 0 0 0 0 1  0  0  0  0  0  0  0  0  0  0  0
## 9  0 0 0 0 0 0 0 0 1 0  0  0  0  0  0  0  0  0  0  0
## 10 0 1 0 0 0 0 0 0 0  0  1  1  1  0  0  0  0  0  0  0
```

```
## 11 0 1 0 0 0 0 0 0 0  1  0  1  1  0  0  0  0  0  0  0
## 12 0 0 0 0 0 0 0 0 0  1  1  0  0  0  0  0  0  0  0  0
## 13 0 0 0 0 0 0 0 0 0  1  1  0  0  0  0  0  0  0  0  0
## 14 0 0 0 1 0 0 0 0 0  0  0  0  0  0  1  1  1  1  1  0
## 15 0 0 0 1 0 0 0 0 0  0  0  0  0  1  0  1  1  0  0  0
## 16 0 0 0 0 0 0 0 0 0  0  0  0  0  1  1  0  1  0  0  0
## 17 0 0 0 0 0 0 0 0 0  0  0  0  0  1  1  1  0  0  0  0
## 18 0 0 0 1 0 0 0 0 0  0  0  0  0  1  0  0  0  0  1  0
## 19 0 0 0 0 0 0 0 0 0  0  0  0  0  1  0  0  0  1  0  0
## 20 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
```

```
library(RColorBrewer)
par(mar=c(1,1,1,1),mfrow=c(2,3))
col5 <- brewer.pal(5, 'Set1')
cols <- ifelse(aprioriformal[[1]] == 1, col5[1],
          ifelse(aprioriformal[[1]] == 2, col5[2],
           ifelse(aprioriformal[[1]] == 3, col5[3],
            ifelse(aprioriformal[[1]] == 4, col5[4], col5[5]))))
par(mar=c(1,1,2,1),mfrow=c(1,1))
plot(snasymmformal, main = 'Apriori Block Model', coord=mycoord,
     vertex.cex =3, edge.col='azure4', vertex.col=cols,
     vertex.border='azure4', label=seq(1:20), label.pos=5,
     label.cex=.5, label.col='gray15')
```



Exploratory block model

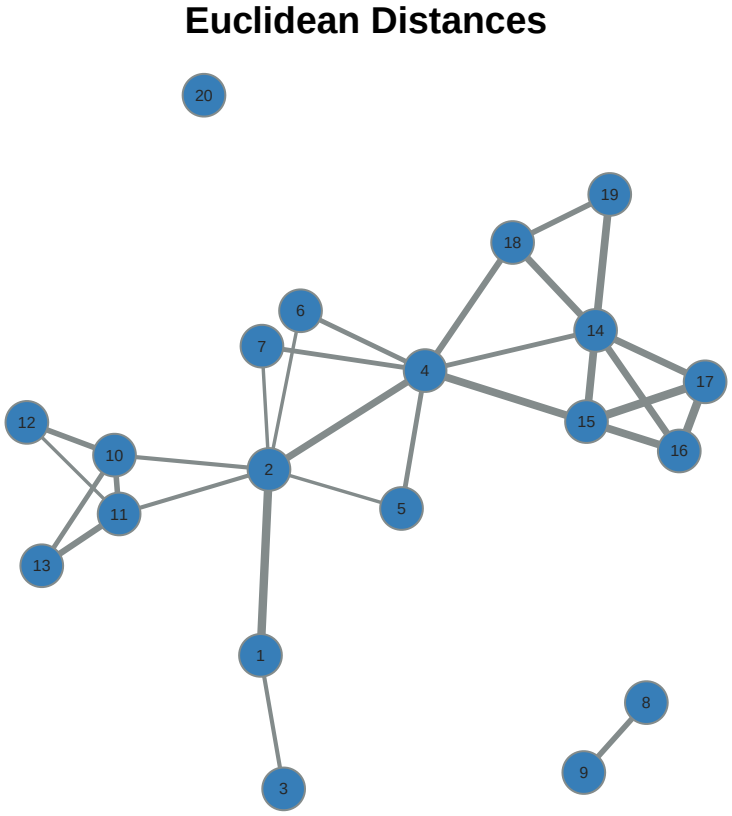
Of course, it would be just great if the roles have always been identified for us, but in real life, it's not going to happen. So we have to have a way to find these roles ourselves, and as we've seen in lecture, there are a few ways to do that. So we are going to use the instruments we have already to try and extract the roles from the network.

Distance Matrix & Hierarchical Clustering

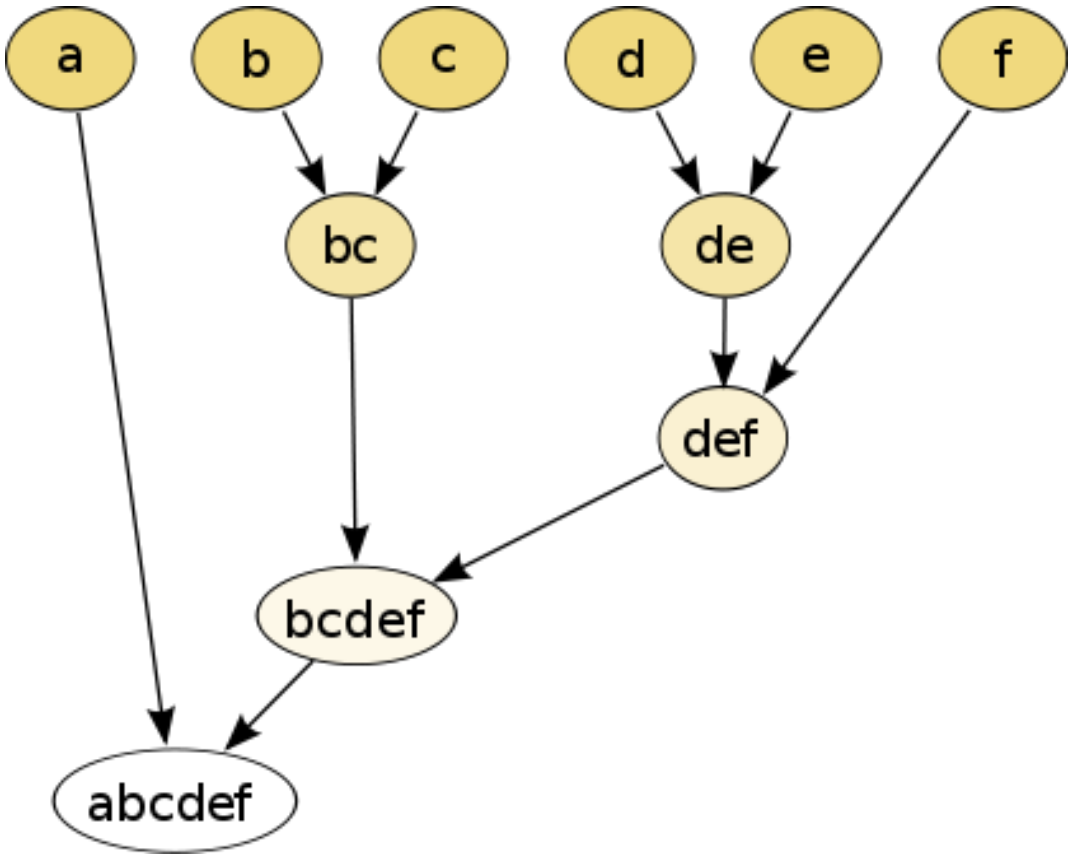
We can use Euclidian distances to create a distance matrix. Remember we created an "or rule" for our data? We'll refer to it when generating euclidian distances:

```
# Create an object of distances in the "OR rule," and turn it into a vector
distformal <- dist(snasymmformal, method="euclidian", diag=FALSE)
thick <- as.vector(distformal)

# Now, let's visualize these distances as edge thickness
par(mar=c(0.5,0,2,0))
plot(snasymmformal, main = 'Euclidean Distances', coord=mycoord,
     vertex.cex =3, edge.col='azure4', vertex.col=col5[2],
     vertex.border='azure4', label=seq(1:20),label.pos=5,
     label.cex=.5,label.col='gray15', edge.lwd = thick^2)
```



What we have to do next is hierarchical clustering, a method of data clustering based not on even clusters, but on clusters as subclusters of larger structures (you may have seen this method in your previous statistical courses). I've shown you the picture of it in the lecture, but here is another example:



Command for doing so is just "hclust," it's an R-native command:

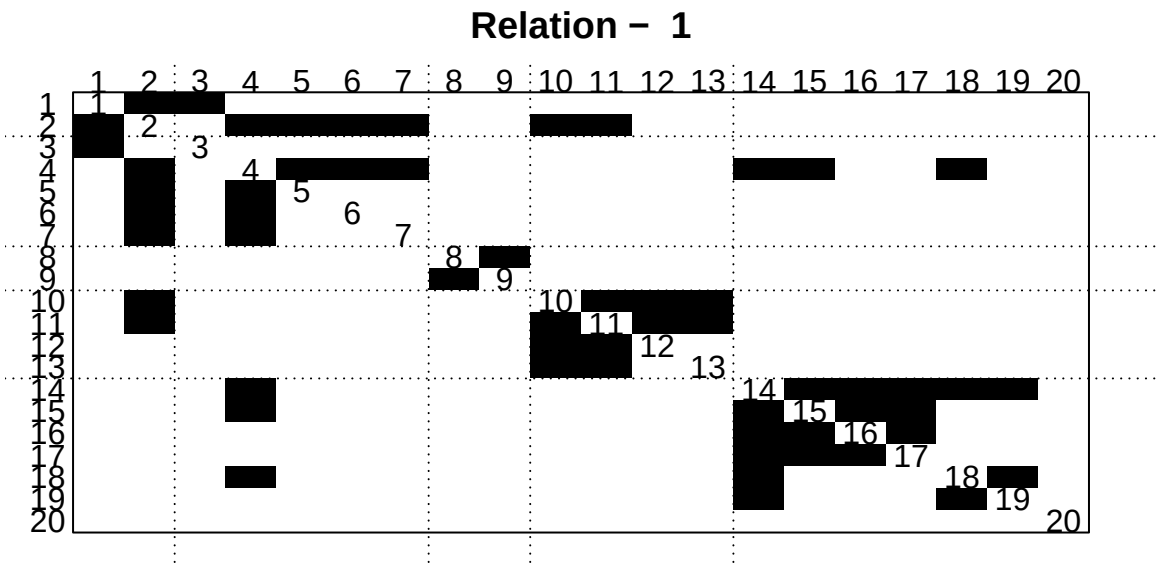
```
# Cluster analysis
formalclust <- hclust(distformal, method="complete")
```

Exploratory blockmodel

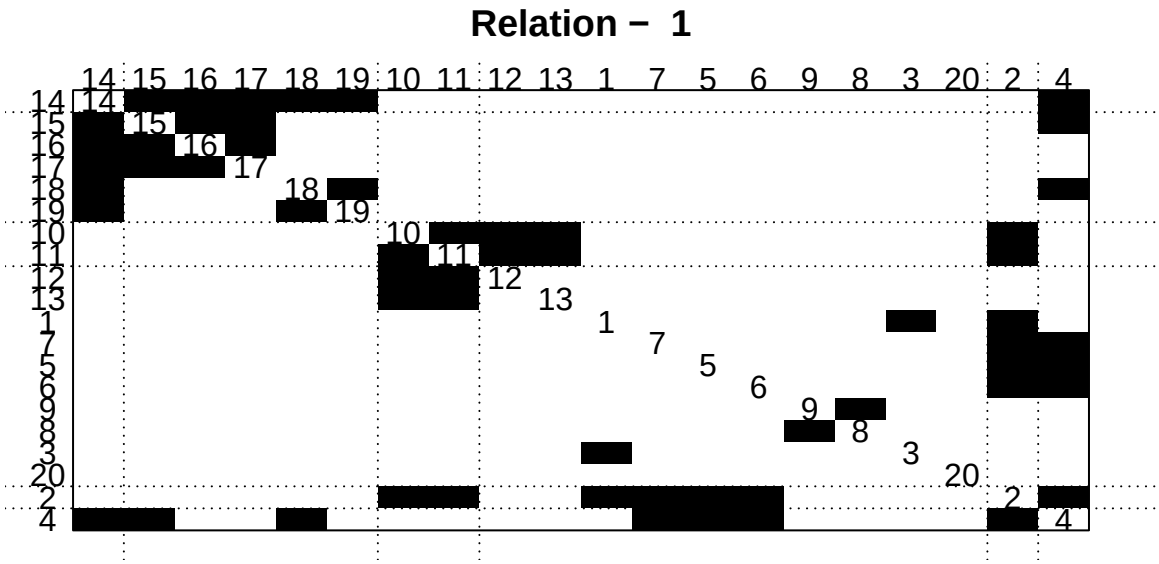
Once we have created a formal set of clusters based on the *hclust* command, we can use clusters to build blockmodels:

```
# And now, a blockmodel based on clustering:
exploratoryformal<-blockmodel(snasymmformal, formalclust, k=6,
                              block.content="density", mode="graph",
                              diag=FALSE)

# Plot the two blockmodels one after another for comparison:
par(mar=c(0,0,2,0))
plot.blockmodel(aprioriformal)
```




```
plot.blockmodel(exploratoryformal)
```

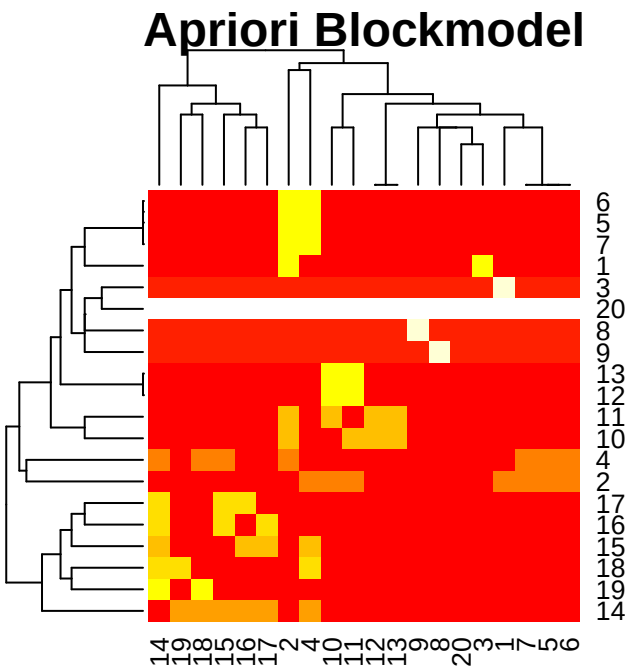


Assignment task.

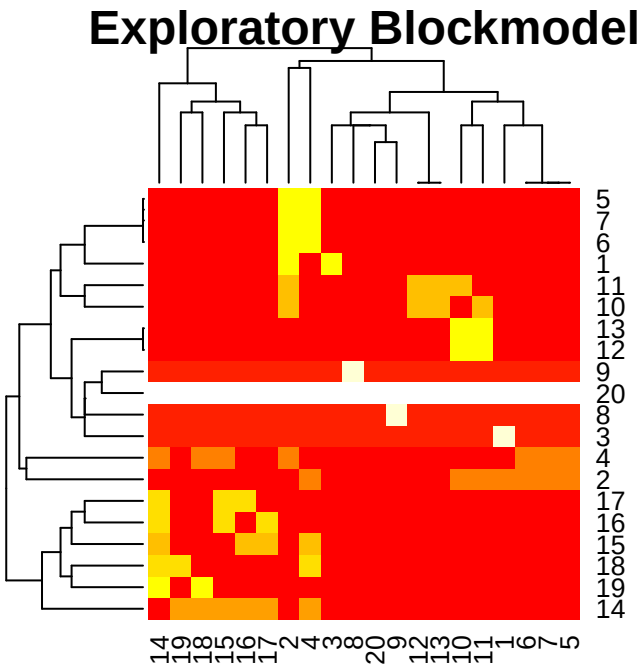
1. Experiment with k. We've set it to 6, but would another number make more sense?
2. Which of the two blockmodels appear to be more accurate to you? Why?

Finally, we can make a heatmap of the two blockmodels:

```
par(mar = c(1,1,4,1), mfrow = c(1,2))
heatmap(aprioriformal[[4]], main ='Apriori Blockmodel')
```




```
heatmap(exploratoryformal[[4]], main ='Exploratory Blockmodel')
```



Blockmodeling based on CONCOR function

You will need to carefully look at the lecture to realize what a CONCOR function does. Remember, this is a method that is based on CONvergence of iterated CORrelations, when we use the similarity between the columns of a correlation matrix to group data together. A very reasonable reference for how it works is provided here: https://www.r-bloggers.com/concor-in-r/, even with the CONCOR routine available as a package. However, without seeing the code of the routine, we can't be certain that it's done correctly, so we are using a function that we can guarantee works correctly (you can check it for yourself - just follow my code, we have put it in the section "For the Most Curious.") The function itself is written below, and you need to copy it entirely into the console and into the RMarkdown, or the method will not work.

When you run your matrix through a CONCOR function, it will generate a set of blocks. The nodes inside them are the closest to each other, but it does not mean the partition is final. They are the closest to each other only *in relation* to the other blocks - meaning, through the iteration of correlations, they've become separated from the rest. But we can - and should - examine each obtained block again (on the original matrix) to make sure that it can't be partitioned further.

The CONCOR function

Let's build the CONCOR function. First, remove the isolate that is not connected to anyone or correlates with anyone (node 20):

```
connectedformal<-formal[-20,-20] # operation on the matrix
class(connectedformal)
```

```
## [1] "matrix"
```

Now, the CONCOR function:

```
CONCOR <- function(mat, max.iter=1000, epsilon=1e-10){
  mat <- rbind(mat, t(mat)) # stack
  colN <- ncol(mat) # width
  X <- matrix(rep(0, times=colN*colN), nrow=colN, ncol=colN)
  target.abs.value <- colN * colN - epsilon # convergence target
  for (iter in 1:max.iter){
    for(i in 1:colN){
      for(j in i:colN){
        X[i,j]<-cor(mat[,i], mat[,j], method=c("pearson"))
      } # end for j
    } # end for i
    mat <- X+(t(X)-diag(diag((X))))
    if (sum(abs(mat)) > target.abs.value) { # test convergence
      #Finished before max.iter iterations
      return(mat)
    } # end if
  } # end for iterations
  return(mat) # return matrix
} # end function
```

Blockmodeling based on CONCOR

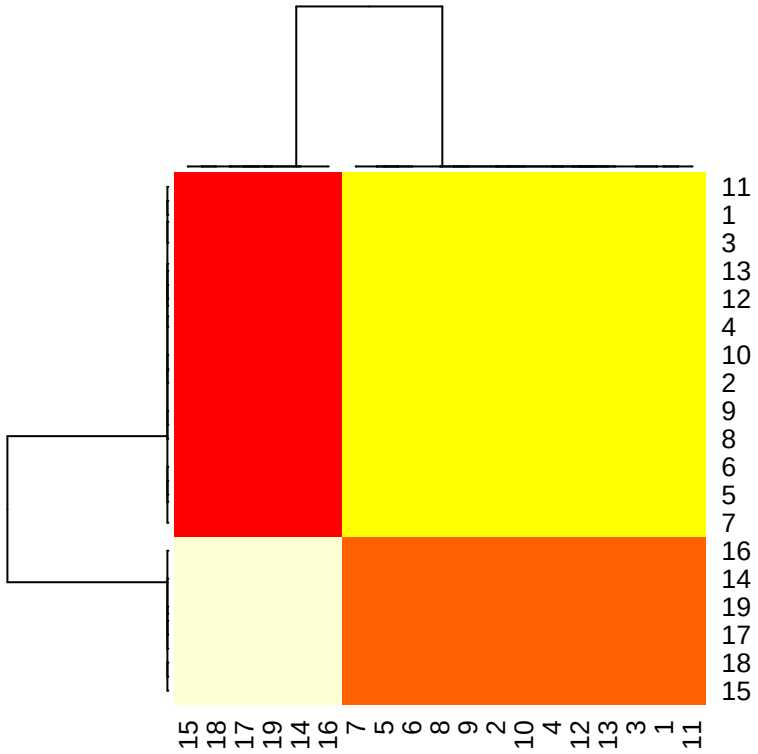
Now, let's create a blockmodel based on CONCOR function. We take the original matrix of data and run it through the function.

```
rownames(connectedformal) <- row.names(roles)[1:19]
colnames(connectedformal) <- row.names(roles)[1:19]

# Command below is commented out, but you can look at it to see what the resulting matrix is.
## connectedformal

#Now, run the matrix through the CONCOR function and show the blockmodel:
CONCORFORMAL<-CONCOR(connectedformal)

# You can look at the matrix on your own; we commented it out to save space in the document:
## print(CONCORFORMAL)
heatmap(CONCORFORMAL)
```

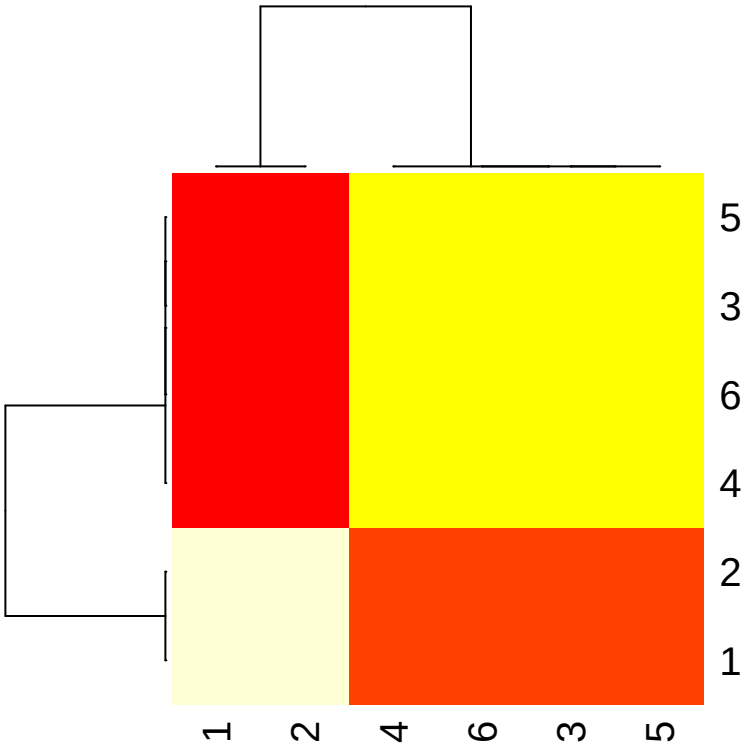


We have obtained the first result. Now, obviously we have four distinct blocks (actually, it's 2x2, so we only need to examine two blocks). Can we separate them further? We should at least try. Are they meaningful? Let's look at them:

```
## part 1  -it's blocks from 14 to 19:
part1 <- connectedformal[14:19,14:19]
colnames(part1) # Who are in this partition?
```

```
## [1] "Ms. Hughes"  "O'Brian"     "Anna"        "Gwen"        "Ms. Patmore"
## [6] "Daisy"
```

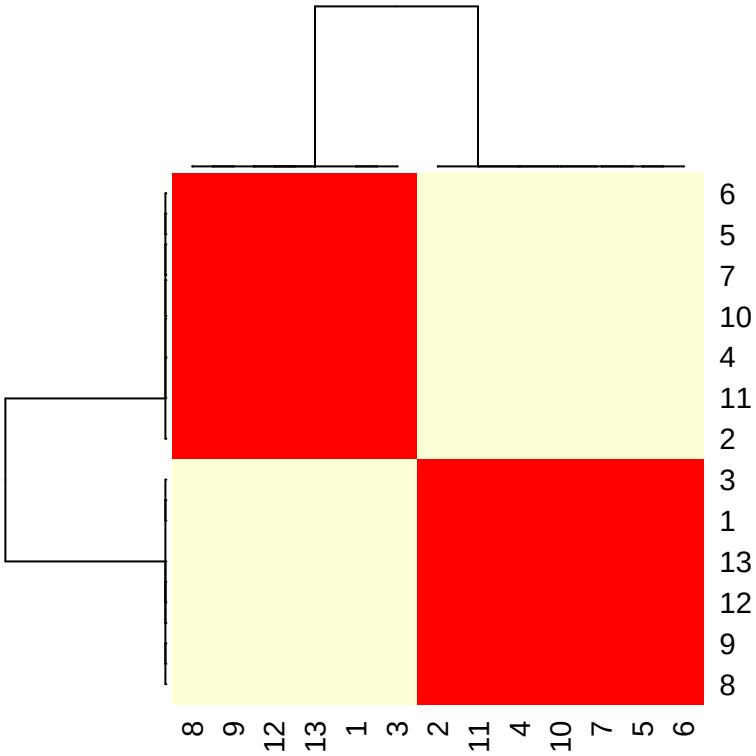
```
concor1 <- CONCOR(part1)
heatmap(concor1)
```



In partition Part11 we have Hughes and O'Brian and Part12 = Anna, Gwen, Ms.Patmore, and Daisy. This partition seems meaninful, given our data. What about the second block? Actually, it's rather large, from 1 to the 13th node. Let's isolate it into a separate matrix and run it through CONCOR again:

```
part2 <- connectedformal[1:13,1:13] # isolate the first 13 nodes

# We commented the matrix out, but you can look at it on your own
##part2
concor2 <- CONCOR(part2) # Run through CONCOR
heatmap(concor2) # Look at the result
```



Looks like we've gotten another set of blocks, which we can look at separately. Let's create a third partition:

```
part3<-c(1,3,8,9,12,13) # isolate the needed nodes
part3.1<-part2[part3,part3] # remove the isolates from partition 2
colnames(part3.1) # Who is here?
```

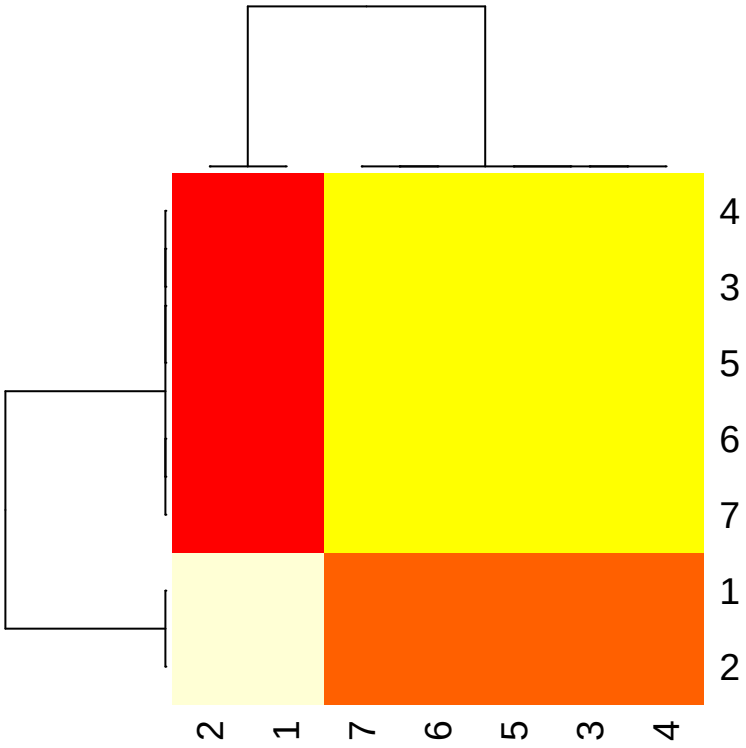
```
## [1] "Violet Grantham"        "Lady Rosamund Grantham"
## [3] "Isobel Crawley"         "Matthew Crawley"
## [5] "Thomas"                 "William"
```

Now, to see if we can partition block 31 further, we need to run it through CONCOR function again. If it does not run, generating an error, it means that the partition we've obtained was final. And it was final - run the code below in CONSOLE ONLY, because it will choke the RMarkdown with errors it generates:

```
## concor3.1<-CONCOR(part3.1)
```

But this is your indication that we do not continue breaking up the matrix - we have found the final partition for section 31. What about section 32? Let's extract what's left of the partition 2 after we removed the first six nodes:

```
part3.2 <- part2[-part3,-part3] # Extract remaining nodes from part2
concor3.2 <- CONCOR(part3.2)  # Run it through CONCOR
heatmap(concor3.2)
```



So unlike part 3.1, part 3.2 could be partitioned again - it ran through CONCOR and we got yet another heatmap. We can look at the two partitions to see if we can further break them up:

```
colnames(part3.2[1:2,1:2]) # Names in the first subpart
```

```
## [1] "Robert Grantham" "Cora Grantham"
```

```
colnames(part3.2[3:7,3:7]) # Names in the second subpart
```

```
## [1] "Lady Mary Grantham"  "Lady Edith Grantham" "Lady Sybil Grantham"
## [4] "Mr. Carson"          "John Bates"
```

Now, in nodes 3-7 we have three Grahams and two random people. But the final attempt at partitioning them will indicate that this block cannot be broken further. There is obviously something in that group of people that makes them connected:

```
part3.2.2 <- part3.2[3:7,3:7] # Create a partition

# Code below will choke RMarkdown, run it in CONSOLE ONLY (it's commented out here):
##concor3.2.2<-CONCOR(part3.2.2)
```

So, as a result, we have broken up our matrix, using the CONCOR function, on a number of blocks.

Assignment task. Try not to get lost in all the partitions! Please list all the finite block-partitions that we have generated and the names of all people that ended up in every block

---

And this is it for today's seminar.

For the most curious: the CONCOR function

This is how the CONCOR function was built and how it works: feel free to explore the code a line at a time.

```
# Set up an example matrix.
mat <- matrix(rbinom(25, 1, 0.5), nr = 5, nc = 5)
colnames(mat) <- c('A','B','C','D','E')
rownames(mat) <- c('A','B','C','D','E')

# Stack the matrix with its transpose
mat
```

```
##   A B C D E
## A 1 1 1 0 0
## B 0 1 1 1 0
## C 1 0 0 1 0
## D 0 1 1 1 0
## E 0 1 0 0 1
```

```
t(mat)
```

```
##   A B C D E
## A 1 0 1 0 0
## B 1 1 0 1 1
## C 1 1 0 1 0
## D 0 1 1 1 0
## E 0 0 0 0 1
```

```
# what if this was a symmetrical matrix? Would it work? -- think about it. :-)
mat <- rbind(mat,t(mat))

# Then concor makes a square matrix of 0s of the same dimensions as mat's width

X <- matrix(rep(0, times=5*5), nrow=5, ncol=5)
X
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0
## [3,]    0    0    0    0    0
## [4,]    0    0    0    0    0
## [5,]    0    0    0    0    0
```

```
# Then for each cell in X it puts the correlation between the stack matrix's
# columm of Xrow and the colum of Xcol
X[2,4]<-cor(mat[,2], mat[,4], method=c("pearson")) ##
X
```

```
##      [,1] [,2] [,3]      [,4] [,5]
## [1,]    0    0    0 0.0000000    0
## [2,]    0    0    0 0.3563483    0
## [3,]    0    0    0 0.0000000    0
## [4,]    0    0    0 0.0000000    0
## [5,]    0    0    0 0.0000000    0
```

```
# Rembember the formula for pearson's r?
## cor(XY) = cov(X,Y) / sd(X)*sd(Y)
# where:
```

```
## cov(X,Y) = E[(X - MuX)*(Y - MuY)]
## and sd(X) = sqrt(E( X - MuX)
###

# The function works until it finishes filling the X matrix once for each cell in X

for(i in 1:5){
  for(j in i:5){
    X[i,j]<-cor(mat[,i], mat[,j], method=c("pearson"))
  } # end for j
} # end for i
X
```

```
##      [,1]       [,2]       [,3]      [,4]        [,5]
## [1,]    1 -0.2182179 -0.2000000 0.0000000 -0.21821789
## [2,]    0  1.0000000  0.2182179 0.3563483 -0.04761905
## [3,]    0  0.0000000  1.0000000 0.0000000 -0.65465367
## [4,]    0  0.0000000  0.0000000 1.0000000 -0.35634832
## [5,]    0  0.0000000  0.0000000 0.0000000  1.00000000
```

```
# Then it makes a stacked matrix again
mat <- X+(t(X)-diag(diag((X))))
mat
```

```
##            [,1]        [,2]       [,3]       [,4]        [,5]
## [1,]  1.0000000 -0.21821789 -0.2000000  0.0000000 -0.21821789
## [2,] -0.2182179  1.00000000  0.2182179  0.3563483 -0.04761905
## [3,] -0.2000000  0.21821789  1.0000000  0.0000000 -0.65465367
## [4,]  0.0000000  0.35634832  0.0000000  1.0000000 -0.35634832
## [5,] -0.2182179 -0.04761905 -0.6546537 -0.3563483  1.00000000
```

```
# Then it iterates with the updated X and the updated mat
# Then it iterates...
# Up to n times or until it only has -1s and 1s in the matrix
```

Homework 3

As we have seen already, homework assignments are somewhat more involved than regular seminars. Unlike previous homeworks, in today's homework we are not showing you new commands - there are plenty of them in the seminar. So we are asking you to apply what you've learned in today's seminar to a new dataset and provide interpretive explanations of your results. You are free to explore as much as possible, but at the very minimum, please do the following:

1. Choose a dataset from one of our previous labs.
2. Apply the same routines we did for the exploratory blockmodel here (it's just copy/paste and then explore the k option). Make a heatmap for your model, vary the k, make a heatmap again. . . ..etc., until you select a k.
3. Apply the CONCOR function to the dataset you selected and plot its heatmap side by side with the heatmap for your exploratory blockmodel.
