Chapter 9: Community Detection of People and Variables (Fisher Data)

kmg

6/16/2021

Contents

The code provided here first runs community detection on *people*. That is, the nodes are people, and they are separated according to similarities in their network patterns.

The second set of code shows how community detection can be used to subset *variables*, much like what we do in P-technique / factor analysis. Results are compared to exploratory factor analysis.

Obtaining Subgroups of Individuals with similar Dynamic Processes

We'll use the same Fisher data from Chapter 6 (on model building) as an exemplar here. We'll create subgroups of people using community detection on their individual dynamic networks (i.e., coefficient estimates in matrix form). This is an option in the GIMME package.

Prepare environment

```
# Install these packages to conduct all analyses in this document.
require(gimme)
require(perturbR)
```

```
require(reshape2)
require(ggplot2)
require(igraph)
require(EGAnet)
require(psych)

# be sure to change the directory to where you put your data!
load("~/Google Drive/Classes/IAV/Data/Fisher/FisherData.Rdata")

# Select a few variables from all individuals.
dataall<- list()
for (p in 1:length(FisherDataInterp)){
  dataall[[p]] <- cbind(FisherDataInterp[[p]]$irritable, FisherDataInterp[[p]]$restless,FisherDataInterp
  colnames(dataall[[p]]) <- c("irritable", "restless", "worried", "guilty", "anhed.", "hopeless", "down
}
```

Identifying subgroups with similar patterns of relations.

The subgroup search first conducts a group-level search to identify which paths exist for the majority.

Before moving to the individual-level search, it clusters individuals based on similarities in the patterns of relations among variables as well as estimates. This is referred to as S-GIMME (see Gates, Lane, Varangis, Giovanello, & Guiskewicz, 2017, for details). Technically, community detection is conducted on a similarity adjacency matrix that quantifies how similar two given individuals are (the edges).

Once individuals are clustered, the algorithm searches for paths that are found for the majority of individuals within each subgroup. The group-level paths are used as a starting point for this search and are again estimated for all individuals. The default in the *gimme* package is that a path is considered to be found for the majority of the subgroup if 51% of the individuals in that subgroup have the path. (This can be altered by setting the argument *subcutoff* to a different value.)

Other options are the community detection method to use (it has been evaluated extensively using Walktrap, which is the default), as well as which features to include to subgroup (the default is to use both the lagged and contemporaneous matrices).

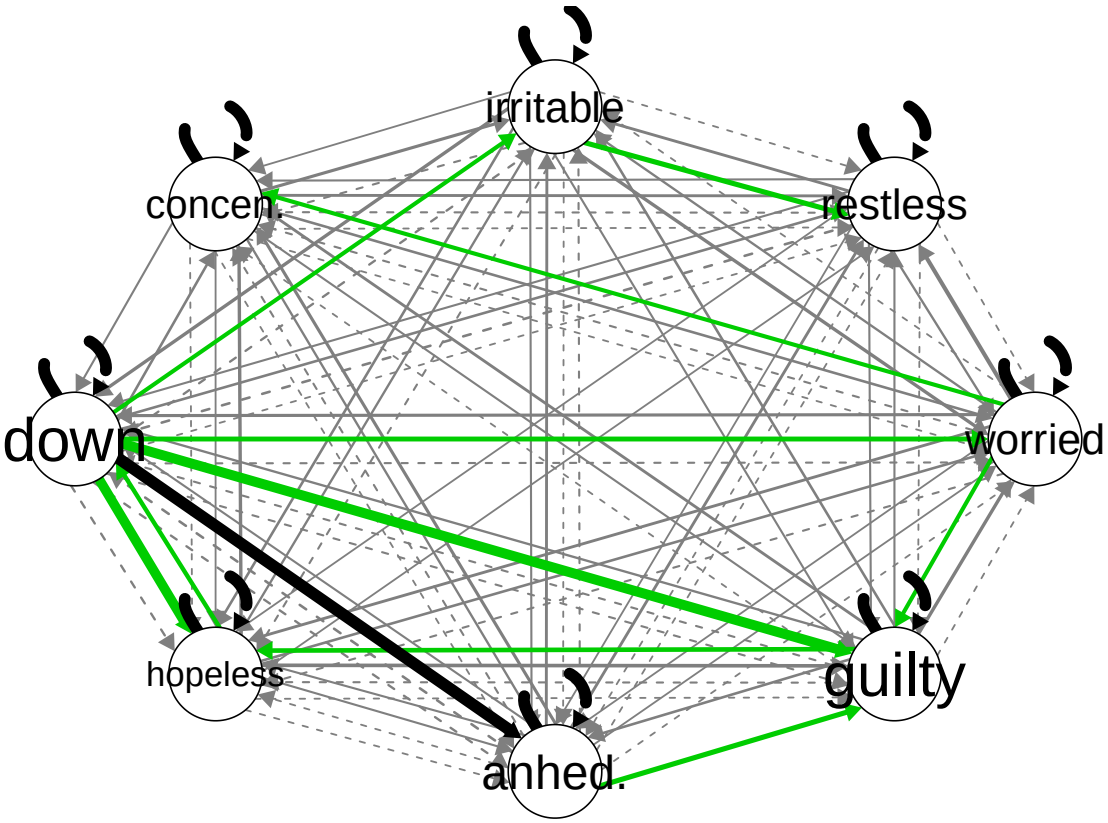
```
outputSubgroup <- gimmeSEM(data = dataall,
                           subgroup = TRUE,
                           sub_method = "Walktrap")
```

```
# see the number of subgroups
length(unique(outputSubgroup$fit$sub_membership))
```

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

```
# plot of aggregated results (note - no aggregation is done during analyses!)
plot(outputSubgroup)
```

```
## Please specify a file id for individual plots. Otherwise, summary plot is presented.
```

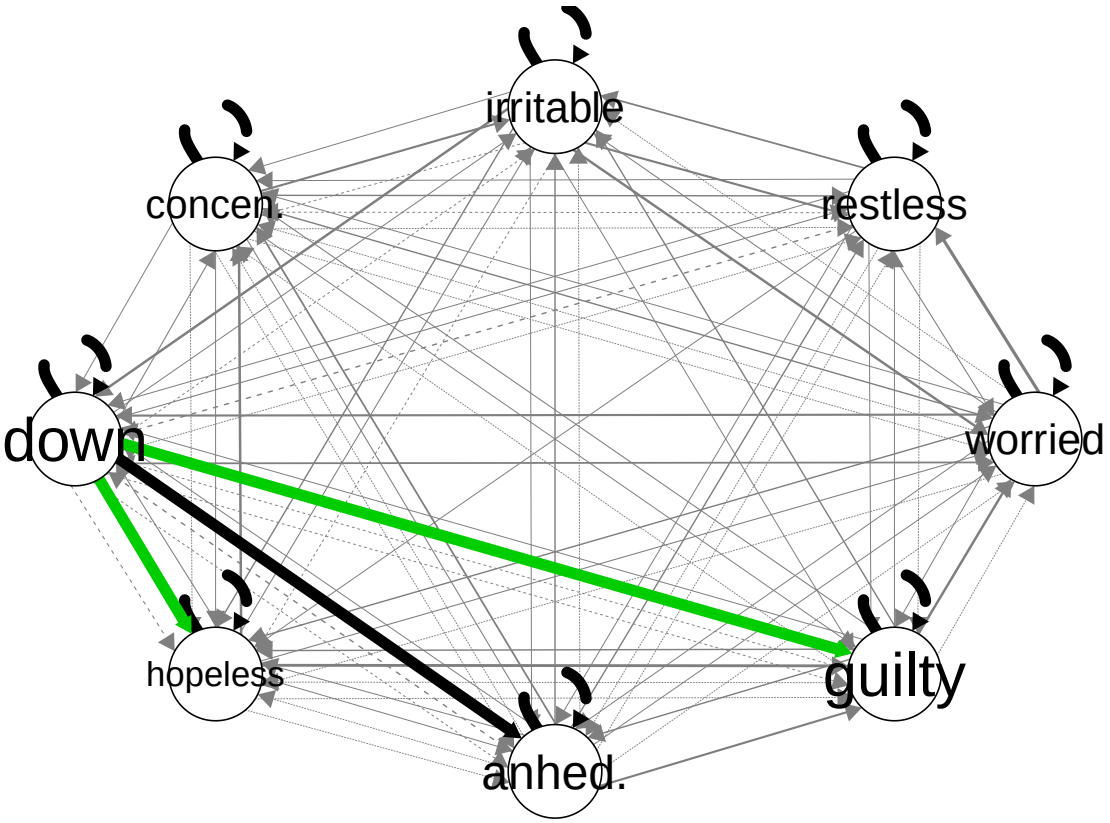


Well this is interesting. We have two subgroups here. A relation between *down* and *anhedonia* emerged the group level (black line), as well as some subgroup-level paths. These are in green.

All relations found at the group and subgroup levels are lag-0, or contemporaneous. We can tell this by the edges being solid. Lag-1 relations are dashed.

Let's take a look at one of the subgroups:

```
# see the number of subgroups
plot(outputSubgroup$sub_plots_paths[[1]])
```



Cool, we see a subgroup-level path between *down* and *hopeless* as well as one between *down* and *guilty*. It seems like, for those in this subgroup, information about how down they are helps to explain their variability in hopeless, anhedonia, and guilty symptoms.

I wonder how many people are in it. We can check.

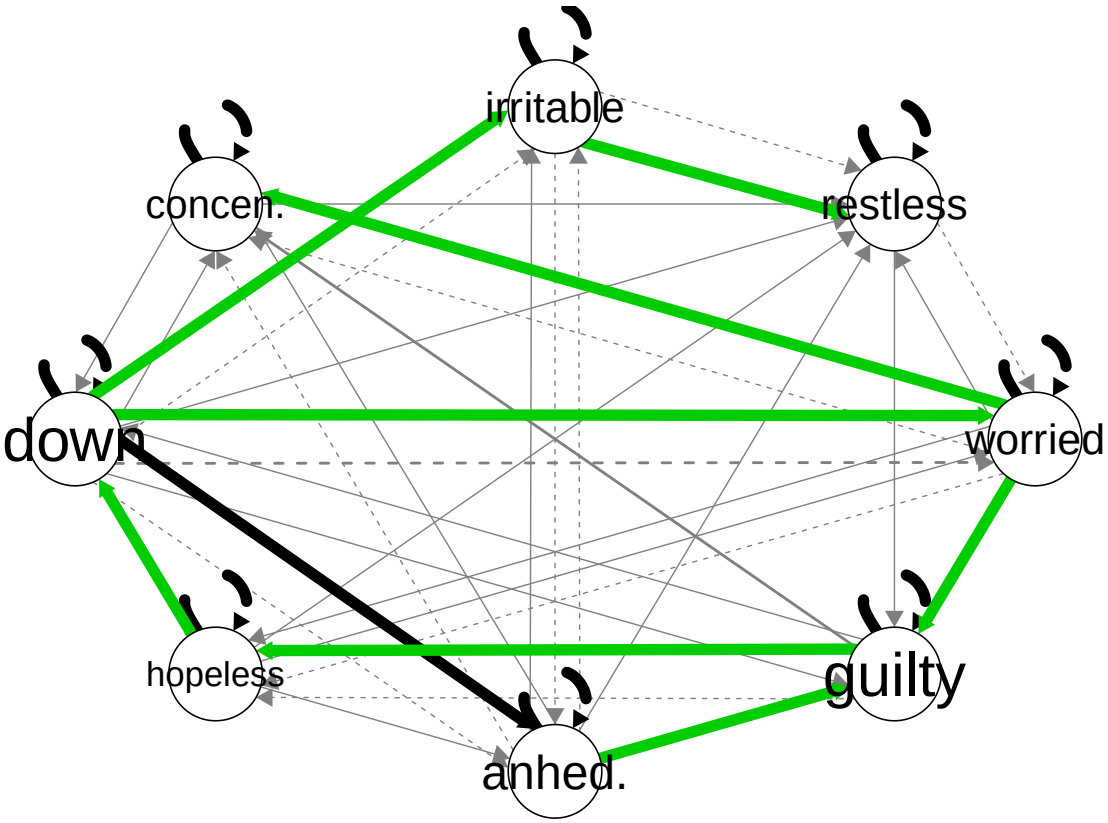
```
length(which(outputSubgroup$fit$sub_membership ==1))
```

```
## [1] 32
```

Thirty-two. So most of the sample!

Let's check out the other subgroup.

```
plot(outputSubgroup$sub_plots_paths[[2]])
```



Wow! A lot o similarities in this subset. So many subgroup-level paths.

Again, *down* seems to explain variability in other variables. However, there is a different symptom set related to down for these individuals. We also see indirect effects between *down* and *restless* (via *irritable*), *concentration* (via *worried*, and *guity* (also via *worried*).

guilty seems like another central node for this subgroup.

View individual results.

Let's look at person 4.

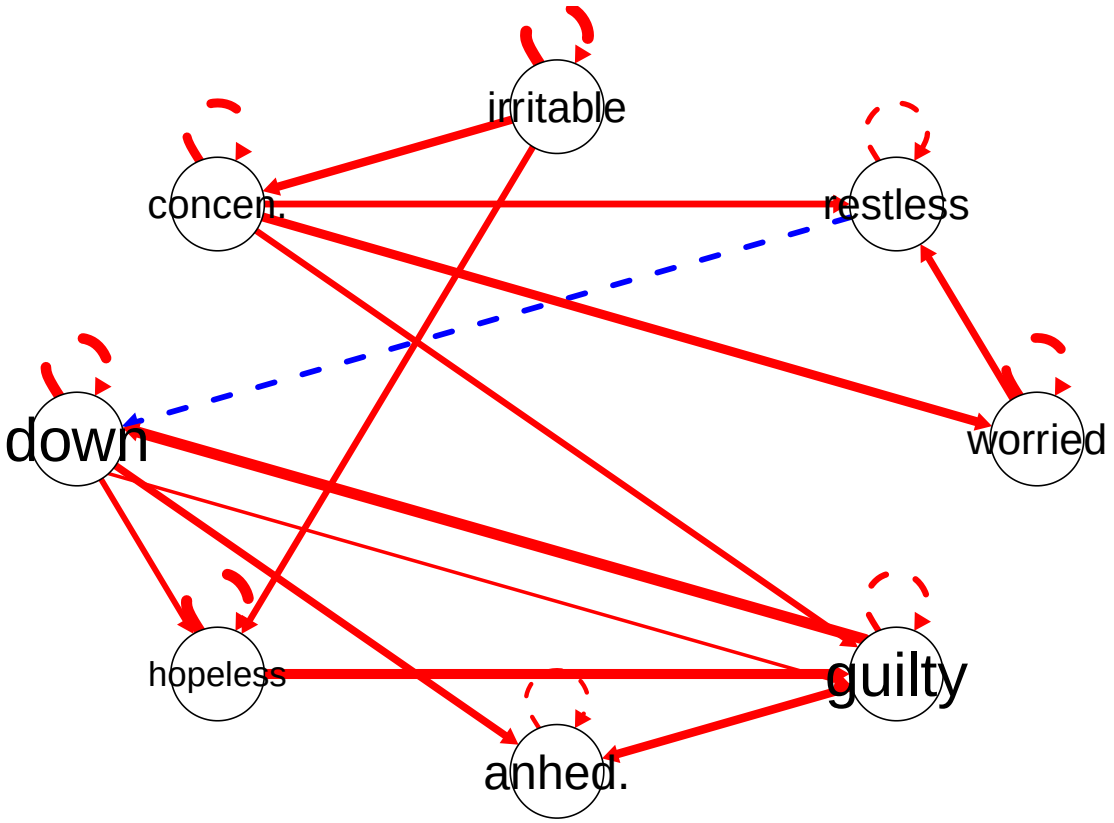
First, let's see what subgroup they belong to.

```
outputSubgroup$fit$sub_membership[4]
```

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

Ok, so they are in the subgroup with fewer subgroup-specific paths.

```
plot(outputSubgroup$plots[[4]])
```



This diagram follows heatmap convention: red = positive (hot); blue = negative (cold).

Evaluating robustness of subgroups

Sometimes you want to know how robust the subgroups are. This is particularly important if your research question is along the lines of, are there two (or more) distinct subgroups that have different processes?

If the question is more along the lines of, "what kinds of relations/edges tend to co-occur for individuals" so you can understand slight variations in temporal proceses that exist for subsets of people, then having robust and highly differentiated subgroups may not be as important.

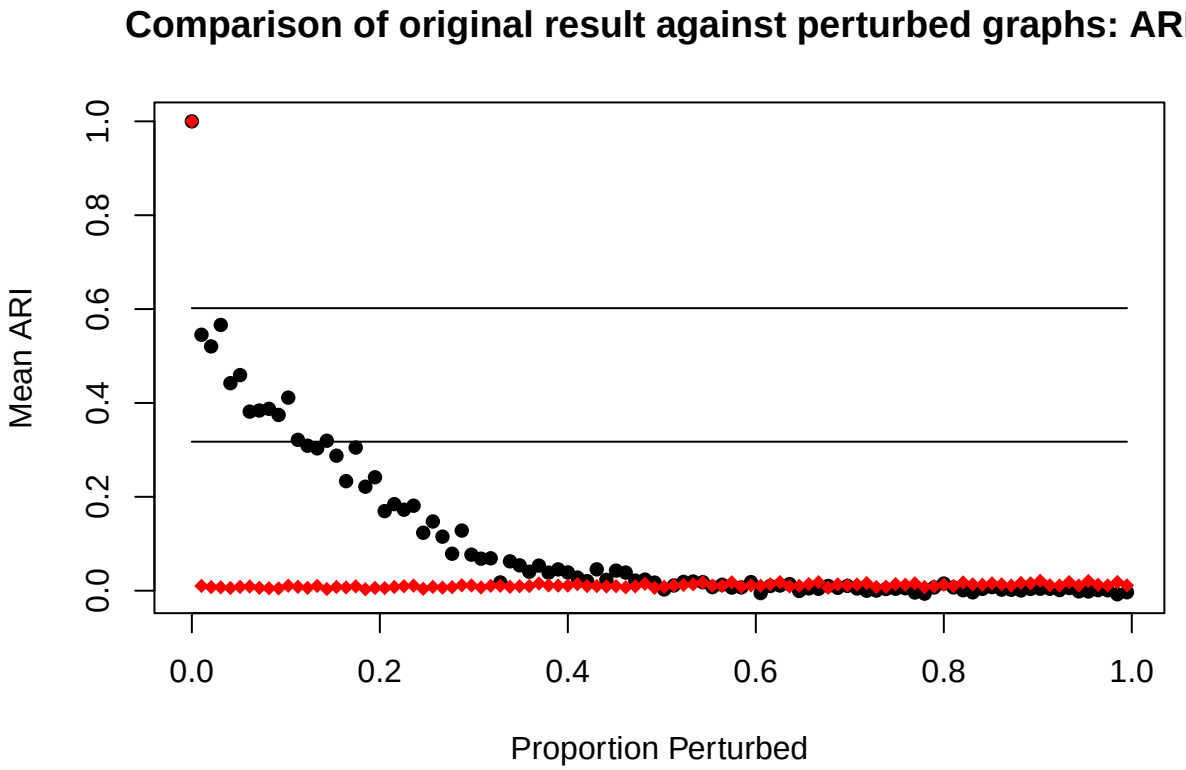
Let's check the solution from the Fisher data and see if the subgroups are robust.

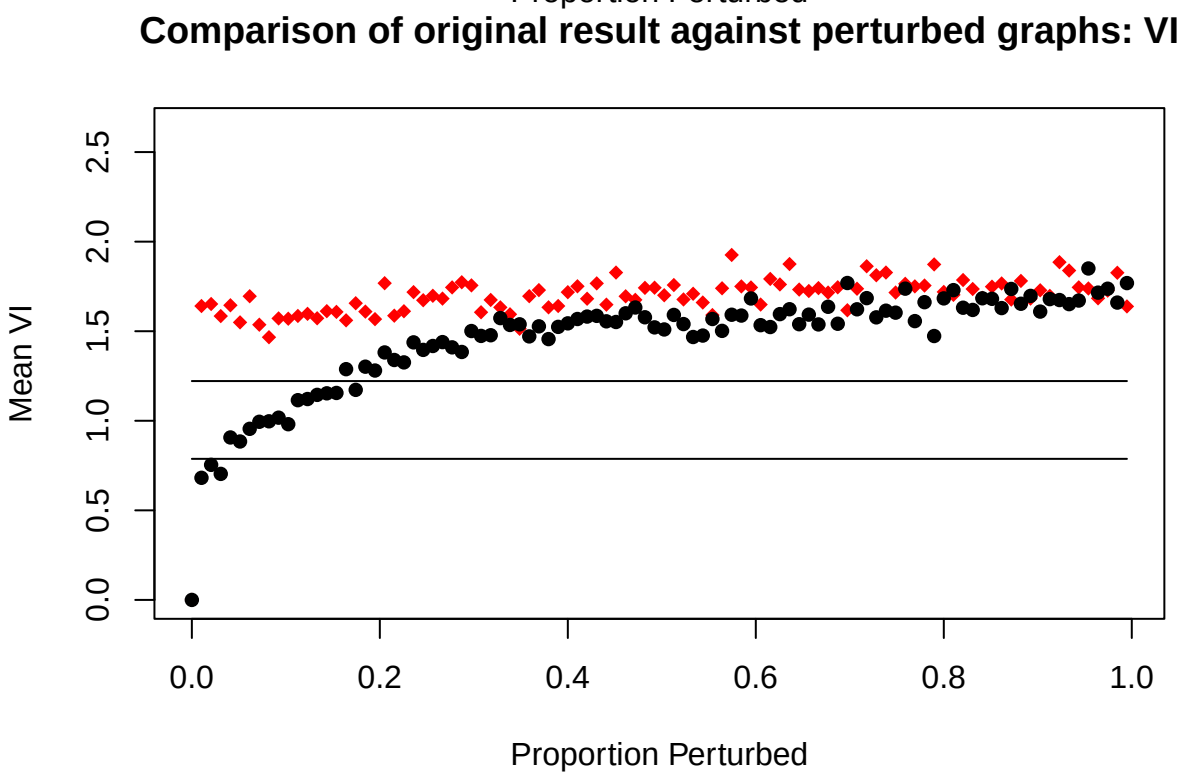
As discussed in Chapter 9, we can evaluate if the solution is robust to minor perturbations by iteratively perturbing the original network, starting with rewiring only 1% of the edges (randomly) until finally we have rewired the entire network, keepign the properties the same as the original network (i.e., the degree distribution for nodes).

Running perturbR

We need to provide the symmetric matrix (sym.matrix) which in our case is a similarity matrix with the nodes being individuals and the edges being a count of how similar two given nodes are. We also can provide information regarding whether we'd like a plot (defaults to true), error bars on that plot (defaults to false), and the number of replications we'd like at each level of rewiring.

```
testRobust <- perturbR(sym.matrix = outputSubgroup$sim_matrix, plot = TRUE, reps = 100)
```





Let's probe the results to make some inferences.

Obtain the VI value when 20% of the community assignments are randomly swapped:

```
testRobust$vi20mark
```

```
## [1] 1.221751
```

Identify the index for the alpha level (percent rewired) for the first time the average VI is greater than this value:

```
min(which(colMeans(testRobust$VI)>testRobust$vi20mark))
```

```
## [1] 17
```

Find alpha that corresponds with this index:

```
testRobust$percent[min(which(colMeans(testRobust$VI)>testRobust$vi20mark))]
```

```
## [1] 0.1641026
```

This suggests that about 17% of the edges need to be rewired for the community detection solution to be as different as if we randomly shifted 20% of the nodes' community assignments. Following convention, that it takes fewer than 20% of edges to be rewired for the solution to be so different suggests that the subgroups may not be robust.

This can be seen both by looking at the point where the black circles intersect with the lower horizontal line in the VI figure and using the code above. A random matrix drops far above this line with only 2% of the edges perturbed in the rewired matrix. The figures provide an immediate evaluation of cluster solutions whereas the code and output allow the user to further investigate the results.

It is also possible to examine the average VI and ARI at the ~20% perturbation point (sometimes the percentages don't fall exactly on 20% given the number of nodes) and see if it is larger than the these marks. Here we provide an example for the ARI values:

```
testRobust$ari20mark
```

```
## [1] 0.3174128
```

```
mean(testRobust$ARI[,which(round(testRobust$percent, digits = 2) == .21)])
```

```
## [1] 0.1695441
```

We see that the distribution of ARI values at $\alpha = 0.21$ is a bit lower than the value obtained when 20% percent of the individuals are randomly assigned a new community.

Taken together, we would probably conclude that this is not a robust community detection solution.

Let's take a look at the modularity values. Is the modularity we obtained higher than expected for random matrice of the same properties?

In this apporach we comppare the modularity values from the original matrix to modularity values obtained from the increasingly perturbed matrices. The output also provides a value called cutoff which is the modularity value that marks the upper 5% percentile in the distribution of modularity results obtained from random matrices that have the same properties as the original matrix.

```
testRobust$cutoff
```

```
## [1] 0.1028751
```

The modularity value from the solution on the original matrix:

```
testRobust$modularity[1,1]
```

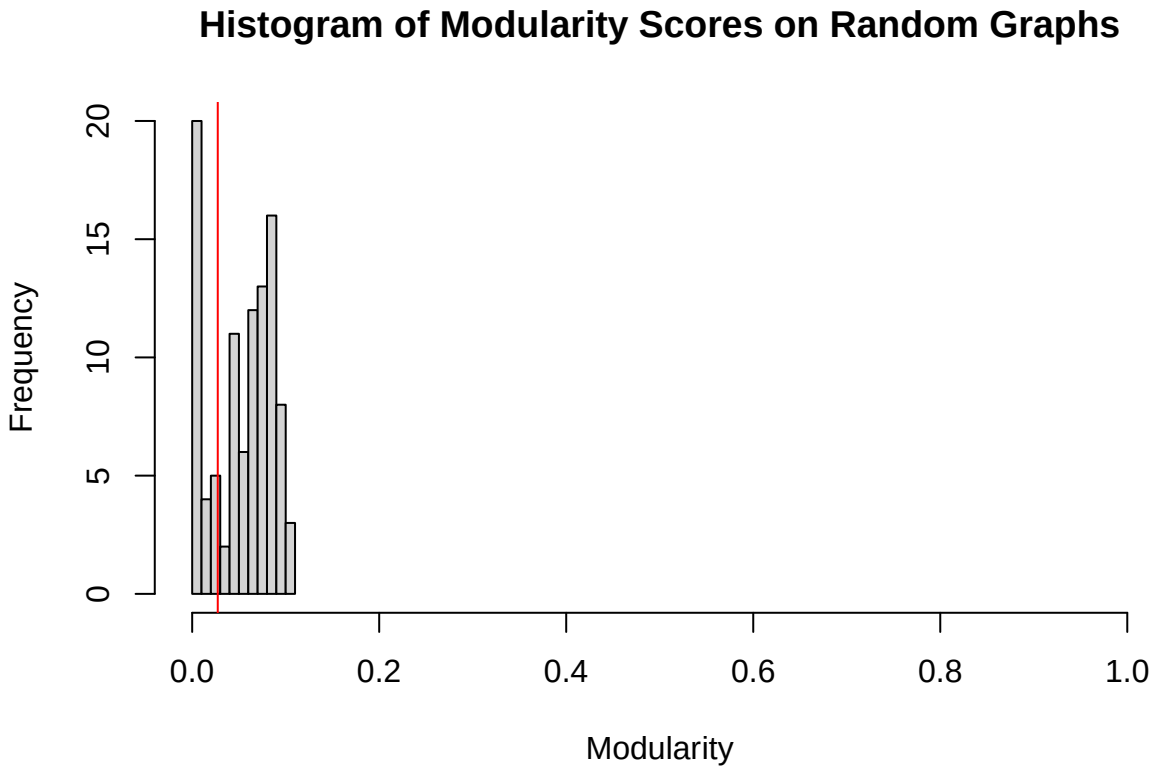
```
## [1] 0.02735434
```

In this example, the cutoff was $Q_{.95} = 0.05$ and the modularity obtained in this simulated data set was $Q_{orig} = 0.044$. (Note that values may differ slightly for each run do to the random generation of matrices.)

Hence the modularity in the original solution was a bit below the upper 5% threshold obtained from the random matrix simulation results.

The figure below depicts a histogram of the modularity values obtained for solutions in the random matrices simulated to have properties similar to the original matrix. Using the code below as a template, researchers can easily obtain similar histograms from the output provided if they would like to explore how the distribution of modularity from the random matrices compares to the modularity obtained in the original matrix.

```
hist(testRobust$modularity[,which(round(testRobust$percent, digits = 2) ==.99)], xlim = c(0,1), main =
abline(v = testRobust$modularity[1,1], col = "red")
```



The red line is the modularity obtained in the observed network solution. If this were a robust solution, it would be higher than the values in the histogram. As it is not, we cannot confidently state that our communities are robustly different from each other as assessed by modularity.

That said, we did identify subsets of individuals with some different dynamic patterns - even if they had some group-level edges constant across the entire group / sample. The results may be meaningful can still be validated with external data to see if the subtypes of processes relate to an outcome or correlate of interest.

Community Detection: Comparison to EFA

Community detection versus Factor Analysis - which is best?? Is this even an appropriate question to ask? Or . . . might they complement each other?

These are all tough questions to answer from a statistical theoretic perspective. Oftentimes your substantive knowledge and what you aim to infer from the analysis will guide whether you conceive of a set of variables as communities or if you consider them as latent factors.

You already know from Lecture 3 how to conduct exploratory factor analysis, and from Lecture 1 you know how to read in Fisher's data. So let's begin.

Read in Fisher data

For this practicum, select a person from Fisher's data.

```
fisher_person <- FisherDataInterp[[2]] # Arbitrarily selected

# Let's just select variables that seem negative - the same ones we used above
fisher_person <-fisher_person[,c(4:7, 9,11,12,16)]

colnames(fisher_person)
```

```
## [1] "irritable"   "restless"    "worried"     "guilty"      "anhedonia"
## [6] "hopeless"    "down"        "concentrate"
```

Conduct an EFA

Select a some variables that are of interest to you, or use them all.

Let's use the *psych* package

```
fit1 <- psych::fa(fisher_person, nfactors = 1, rotate = "varimax", fm = "ml")

fit1$RMSEA
```

```
##      RMSEA      lower      upper confidence
##  0.2277704  0.1936133  0.2659037  0.9000000
```

```
fit1$EBIC
```

```
## [1] 230.9077
```

```
fit1$TLI
```

```
## [1] 0.3631546
```

```
fit2 <- psych::fa(fisher_person, nfactors = 2, rotate = "varimax", fm = "ml")

fit2$RMSEA
```

```
##      RMSEA      lower      upper confidence
##  0.0000000  0.0000000  0.0881575  0.9000000
```

```
fit2$EBIC
```

```
## [1] -53.03366
```

```
fit2$TLI
```

```
## [1] 1.007634
```

```
fit3 <- psych::fa(fisher_person, nfactors = 3, rotate = "varimax", fm = "ml")

fit3$RMSEA
```

```
##      RMSEA      lower      upper confidence
## 0.01522478 0.00000000 0.11846736 0.90000000
```

```
fit3$EBIC
```

```
## [1] -28.16042
```

```
fit3$TLI
```

```
## [1] 0.9961375
```

For this exemplar person, there is evidence of a 2 factor solution. The EBIC is lowest here, and the TLI and RMSEA both suggests a good fit.

Let's try an oblique rotation and check out the factor loading structure.

printing factor loading matrix

```
fit2ob <- psych::fa(fisher_person, nfactors = 2, rotate = "promax", fm = "ml")
```

```
## Loading required namespace: GPArotation
```

```
fit2ob$RMSEA
```

```
##      RMSEA      lower      upper confidence
##  0.0000000  0.0000000  0.0881575  0.9000000
```

```
fit2ob$EBIC
```

```
## [1] -53.03366
```

```
fit2ob$TLI
```

```
## [1] 1.007634
```

```
# Here we print just the loadings. To get all the estimates, use print.psych(fit2ob)
print(fit2ob$loadings)
```

```
##
## Loadings:
##             ML1    ML2
## irritable           0.724
## restless            0.767
## worried     0.132  0.699
## guilty      0.582  0.100
## anhedonia   0.559 -0.187
## hopeless    0.648  0.164
## down        1.003
## concentrate -0.104  0.509
##
##                  ML1   ML2
## SS loadings    2.107 1.936
## Proportion Var 0.263 0.242
## Cumulative Var 0.263 0.505
```

For this person, It seems that Guilty, Anhedonia, Hopeless, and Down load on factor one (with Down perhaps being an ideal scaling indicator).

Irritable, Restless, Worried, and Concentrate load on factor 2.

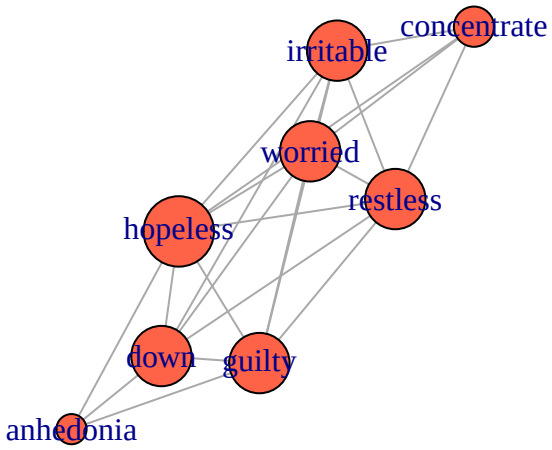
Conduct community detection on the correlation matrices

Here we use Walktrap. Another good option in *igraph* is Label Propagation.

```r
# generate a correlation matrix from the variables you used in EFA
sym.matrixg                  <- as.data.frame(cor(fisher_person, use = "complete.obs"))
# set diagonal to zero, otherwise it's the highest value in the matrix
diag(sym.matrixg)            <- 0

# set negatives to zero
sym.matrixg[sym.matrixg < 0] <- 0

# let's see what the network looks like now:
sym.matrixg
```

```
##              irritable   restless   worried    guilty anhedonia     hopeless
## irritable   0.00000000 0.55723383 0.5144226 0.1590056 0.0000000 0.1933796054
## restless    0.55723383 0.00000000 0.5432278 0.2004262 0.0000000 0.2116902086
## worried     0.51442264 0.54322783 0.0000000 0.1397826 0.0000000 0.2768246576
## guilty      0.15900558 0.20042620 0.1397826 0.0000000 0.3005182 0.3613345129
## anhedonia   0.00000000 0.00000000 0.0000000 0.3005182 0.0000000 0.3828779556
## hopeless    0.19337961 0.21169021 0.2768247 0.3613345 0.3828780 0.0000000000
## down        0.03559155 0.08648115 0.1745611 0.5863413 0.5446974 0.6547264352
## concentrate 0.35824196 0.38428001 0.3716768 0.0000000 0.0000000 0.0004599964
##                   down  concentrate
## irritable   0.03559155 0.3582419595
## restless    0.08648115 0.3842800134
## worried     0.17456114 0.3716767901
## guilty      0.58634131 0.0000000000
## anhedonia   0.54469739 0.0000000000
## hopeless    0.65472644 0.0004599964
## down        0.00000000 0.0000000000
## concentrate 0.00000000 0.0000000000
```

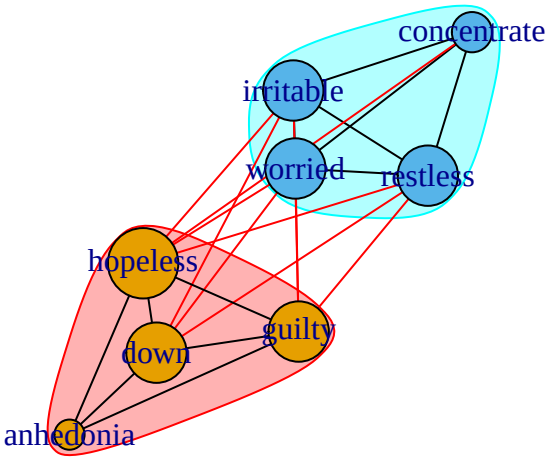
```r
# create a graph object
g                            <- graph.adjacency(as.matrix(sym.matrixg),
                            mode ="undirected", weighted = TRUE)

# Compute node degrees (#links) and use that to set node size:
deg <- degree(g, mode="all")
V(g)$size <- deg*5

plot(g, vertex.color="tomato")
```



```
# conduct Walktrap and safe the variable membership vector
walk <- walktrap.community(g, weights = E(g)$weight, steps = 4)
membership <- walk$membership

cbind(colnames(fisher_person), membership)
```

```
##                    membership
## [1,] "irritable"   "2"
## [2,] "restless"    "2"
## [3,] "worried"     "2"
## [4,] "guilty"      "1"
## [5,] "anhedonia"   "1"
## [6,] "hopeless"    "1"
## [7,] "down"        "1"
## [8,] "concentrate" "2"
```

```
plot(walk, g)
```



We see here that Guilty, Anhedonia, Hopeless, and Down were all found to be in the same community, similar to the factor analytic results where these all loaded on the same factor. Irritable, Restless, Worried, and Concetrate were all found to be in the same community, again matching up with the factor analytic solution.

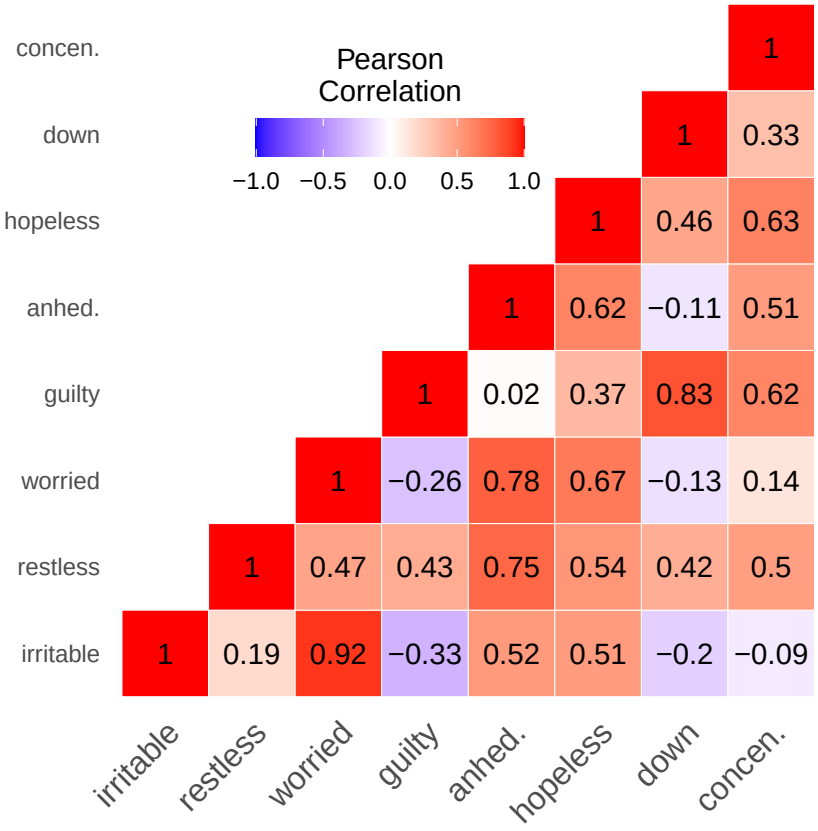
In this case, inferences from results directly match the EFA results. Try a different person. What happens?

It is definitely not always the case that results from community detection align so well with EFA results.

create a heatmap of correlation matrix

```
# Get upper triangle of the correlation matrix
get_upper_tri <- function(cormat){
     cormat[lower.tri(cormat)]<- NA
     return(cormat)
 }
upper_tri <- get_upper_tri(round(cor(outputSubgroup$data[[2]][9:16,9:16]), digits = 2))

melted_cormat <- melt(upper_tri, na.rm = TRUE)
ggheatmap <- ggplot(melted_cormat, aes(Var2, Var1, fill = value))+
     geom_tile(color = "white")+
     scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                          midpoint = 0, limit = c(-1,1), space = "Lab",
```

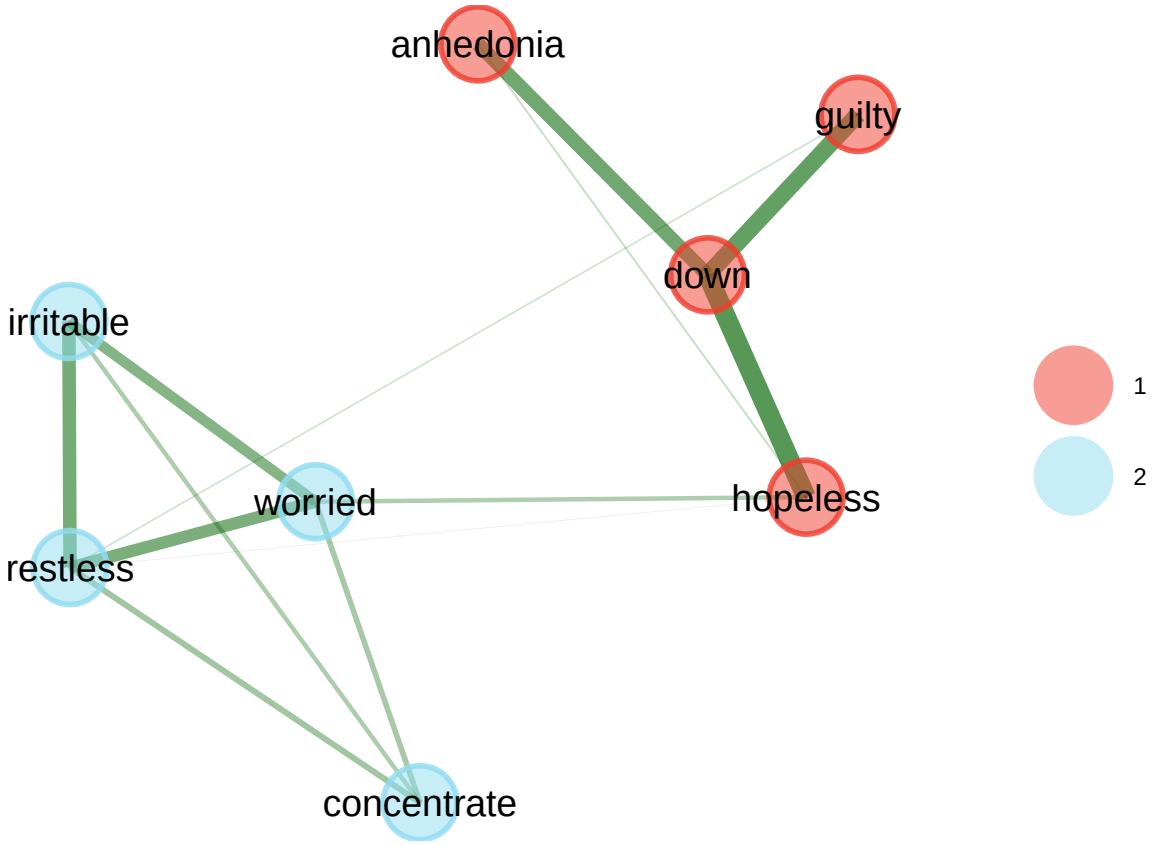
```
                          name="Pearson\nCorrelation") +
    theme_minimal()+ # minimal theme
    theme(axis.text.x = element_text(angle = 45, vjust = 1,
                                     size = 12, hjust = 1))+
    coord_fixed()

ggheatmap +
    geom_text(aes(Var2, Var1, label = value), color = "black", size = 4) +
    theme(
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        panel.grid.major = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        axis.ticks = element_blank(),
        legend.justification = c(1, 0),
        legend.position = c(0.6, 0.7),
        legend.direction = "horizontal")+
    guides(fill = guide_colorbar(barwidth = 7, barheight = 1,
                                 title.position = "top", title.hjust = 0.5))
```



Conduct EGA

Exploratory Graph Analysis (EGA) is very similar to the above. In EGA, prior to conducting community detection, sparsity is induced in the correlation matric. The default approach is GLASSO, which penalizes certain parameters and sets them to zero. We see similar results using EGA.

```r
# get correlation matrix - this time, keep the diagonals. The algorithm sets them to zero.
sym.matrix1                  <- as.data.frame(cor(fisher_person, use = "complete.obs"))

outega <- EGA(sym.matrix1, n = 90)
```




```r
# let's see what was set to zero in the matrix:
outega$network
```

```
##              irritable    restless    worried    guilty  anhedonia    hopeless
## irritable   0.00000000 0.297163129 0.23199645 0.0000000 0.00000000 0.000000000
## restless    0.29716313 0.000000000 0.26898387 0.0370846 0.00000000 0.007484737
## worried     0.23199645 0.268983866 0.00000000 0.0000000 0.00000000 0.098705246
## guilty      0.00000000 0.037084604 0.00000000 0.0000000 0.00000000 0.000000000
## anhedonia   0.00000000 0.000000000 0.00000000 0.0000000 0.00000000 0.044128635
## hopeless    0.00000000 0.007484737 0.09870525 0.0000000 0.04412863 0.000000000
## down        0.00000000 0.000000000 0.00000000 0.3736347 0.31312597 0.433100534
## concentrate 0.09966158 0.132323059 0.12218272 0.0000000 0.00000000 0.000000000
##                  down concentrate
## irritable   0.0000000  0.09966158
## restless    0.0000000  0.13232306
## worried     0.0000000  0.12218272
## guilty      0.3736347  0.00000000
## anhedonia   0.3131260  0.00000000
## hopeless    0.4331005  0.00000000
## down        0.0000000  0.00000000
## concentrate 0.0000000  0.00000000
```