National Research University Higher School of Economics

Faculty of Social Sciences

Introduction to Network Analysis

Seminar 4 and Homework 2: Models of Social Influence

Contents

Valentina Kuskova, PhD

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

Alright, let's try real statistical network models. We start with ERGM.

Contents of today's seminar:

- In your Seminar folder, you will find the following files:
  1. This file in .pdf format, the .rmd format, and all the prep files needed.
  2. A set of articles to help you understand ERGM models better:

     "Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects"

     "ergm: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks"

     The legacy article on fitting p-star models, written by Stan Wasserman ages ago - for mathematical and historical reference.
- 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, RColorBrewer, NetData, network, sna, igraph - these you know
  2. statnet, coda, ergm, rgl - these are new

  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. Learn how to create, fit, and evaluate p-star models.
  2. Put together network and non-network data into models that you can analyze and interpret.
  3. Compare and interpret obtained network results.

Seminar 4 assignment. Please answer the questions that are marked as *Assignment questions.* All previous submission rules apply.

Homework 2 assignment is provided at the end of this document.

Both assignments are due on March 2nd by 23:59. We have a two-week break in schedule because of the holiday, so please use this time wisely to work with your network models.

As usual, 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.

Logistic regression

P-star uses the same general approach as logistic regression, because in the very general sense, we are looking at a probability of making a tie - our dependent variable is dichotomous. In case you need a refresher, we have found a few references for you to brush up on the basics.

1. An intro to logistic regression: http://www.appstate.edu/~whiteheadjc/service/logit/intro.htm
2. Logistic regression for non-statistical audiences: http://www.theanalysisfactor.com/explaining-logistic-regression/
3. Logistic regression for dummies: https://codesachin.wordpress.com/2015/08/16/logistic-regression-for-dummies/
4. Logistic regression in R: http://www.r-tutor.com/elementary-statistics/logistic-regression

Of course, you are welcome to use any other references you wish, including those provided to you in your previous courses at HSE.

Assignment task

You should be able to easily answer the following questions:

1. What is an odds function?
2. What is an odds ratio?
3. What is a log-odds function?
4. What is a logistic function?
5. What is the relationship between predicted probabilities and predicted odds?
6. What is a pseudo-R-square?
7. How do you interpret the results of the logistic regression?

ERGM

ERGM is the name for the family of Exponential Random Graph Models - models that allow us to compare our networks to random networks, with a set of interesting implications. In this seminar, I do not have the time to explain all details about the model itself in details - I only give you the very basics. For details, I refer you to our Lecture and a set of papers that I have put in you Seminar 8 folder - they are relatively easy to read and grasp. In seminars we will only go over the details of the code, and I will (quickly) run you through interpretation.

ERGM Basics

We will use the package, *ergm*, and you have to use it together with several built-in commands that we know already:

1. ergm() - the estimation itself;
2. simulate() - similate a potentially final model;
3. summary() - obtain measurements of network statistics on a dataset or a model.

Model specification is complicated. Each term is a hypothesis and it takes time/effort to learn what terms work for what types of a network and how to interpret them.

A brief taxonomy of terms

1. Basic structural terms: these terms control the overall probability of tie formation. The most basic network structural characteristics we want to account for are density (edges) and reciprocity (in directed networks).
2. Nodal attribute main effects: these are the usual individual level predictors we are used to working with. Inclusion tests the association between one's value in the attribute and the probability of tie

formation. Just as in linear regression, we should be careful about how we code these terms - centering can often facilitate interpretation and we need to avoid empty cells with categorical variables.
3. Interactions for attribute-based mixing: these are the terms we use to test hypotheses about assortative/dissortative mixing.
4. Relational attributes: these are terms for characteristics of dyads and edges instead of individual notes.
5. More complex structural terms: these terms describe structural features of the network (beyond density and dyads), includes terms for triangles/ transitivity, degree distributions, k-stars, isolates, 2-paths, cycles, and shared-partner distributions.
6. Actor-specific effects - receiver, sender, sociality.
7. Whole network operators - I think these are used for simulations only.
8. Constraint terms - these account for particular features of your network that are known a priori.

Workflow

How do you work through setting up a model? Actually, there is a rather straightforward procedure, just as with regular regression:

1. Examine your data: plot the network, look at attributes.
2. Formulate hypotheses to test.
3. Estimate unconditional model (edges)
4. Add mutuality if graph is directed
5. Check for model degeneracy often with *mcmc.diagnostics( )*
6. Add node attribute effects and interactions if needed to test your hypotheses
7. Add more complicated structural terms
8. What other terms are needed to test your hypotheses?
9. When you have a candidate model, simulate "goodness of fit" with *gof()*

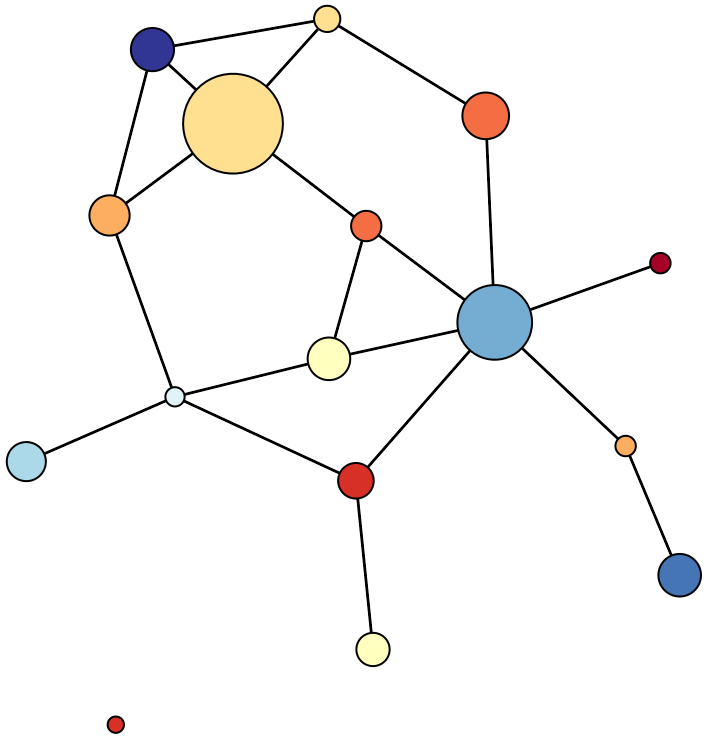
Before you begin, install the necessary libraries. By now, you know how to install them (in Console), so I just call them from here for us to work with:

```
suppressPackageStartupMessages(library(ergm))
suppressPackageStartupMessages(library(coda))
suppressPackageStartupMessages(library(RColorBrewer))
suppressPackageStartupMessages(library(rgl))
suppressPackageStartupMessages(library(sna))
```

The simplest model - the Unconditional

The simplest model is the Bernoulli model (coin flip), which contains only an edge term. We are going to start with our favorite Florentine families:

```
data(florentine, package='ergm')

# Of course, let's look at data first.
# We've seen this image in 3-D a couple of weeks ago:
col20 <- c(brewer.pal(11,'RdYlBu'),brewer.pal(9,'RdYlBu'))
par(mar=c(1,1,1,1))
wealth <- flomarriage %v% 'wealth' # the %v% extracts vertex
plot(flomarriage, vertex.cex = wealth/25+1, vertex.col = col20) #
```



```
#Also, let's look at couple of network statistics, we'll need them later for comparison:
network.density(flomarriage)
```

```
## [1] 0.1666667
```

```
gtrans(flomarriage, mode = 'graph')
```

```
## [1] 0.1914894
```

You will need to know your logistic regression to be able to easily operate with terms shown here. However, I will do all I can to help.

The ergm model applies a logit link to model the outcome, Y_{ij} , which takes the value of 1 if a tie is present between i and j and 0 if it's absent. We start first with the coefficient of *edges* - the model's estimate of the log-odds of any tie occuring in a graph.

```
# Command is simply ergm.
# "edges" means we are adding edges.
flomodel.01 <- ergm(flomarriage ~ edges)
```

```
## Starting maximum pseudolikelihood estimation (MPLE):
```

```
## Evaluating the predictor and response matrix.
```

```
## Maximizing the pseudolikelihood.
```

```
## Finished MPLE.
```

```
## Stopping at the initial estimate.
```

```
## Evaluating log-likelihood at the estimate.
```

```
summary(flomodel.01)
```

```
## Call:
## ergm(formula = flomarriage ~ edges)
##
## Iterations:  5 out of 20
##
## Monte Carlo MLE Results:
##       Estimate Std. Error MCMC % z value Pr(>|z|)
## edges  -1.6094     0.2449      0  -6.571   <1e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 166.4  on 120  degrees of freedom
##  Residual Deviance: 108.1  on 119  degrees of freedom
##
## AIC: 110.1    BIC: 112.9    (Smaller is better.)
```

So how to interpret this model? Notice that just as with any regression, you have a parameter estimate (regression coefficient) and a standard error. Unlike the traditional regression output, however, you do not have the t-statistic (though you still get the p-value and the corresponding stars to indicate model significance). So far, so good - looks like the standard logistic regression model we know.

What about the regression coefficients? Here we refer to logistic regression interpretation. The log-odds of any tie occuring is:

$$-1.6094 \text{ x change in the number of ties}$$

$$= -1.6094 \text{ x } 1,$$

for all ties, since the addition of any tie to the network changes the number of ties by 1!

Corresponding probability is:

$$\frac{exp(-1.6094)}{1+exp(-1.6094)} = 0.1667$$

which is what you would expect, sicne there are 20/120 ties.

This is the same as transforming log-odds into probabilities:

$$logit^{-1}(-1.6094) = \frac{1}{1+e^{-(-1.6094)}}$$

and we can create a simple R function to do so:

invlogit <- function(x){ 1/(1 + exp(-x))g}, where x is our regression coefficient:

```
invlogit <- function(x) {1/(1 + exp(-x))}
x<-coef(flomodel.01)
invlogit(x)
```

```
##     edges
## 0.1666667
```

Pay attention: what does the probability equal to? Exactly our network density.

A model conditional on triangles

The simplest model is not very interesting - it does not give us much beyond the simple probability of forming an edge on an existing network. To move from that, we can add a term often thought to be a measure of "clustering" - the number of completed triangles. Because now we get into stochastic simulation, your output

will differ. The term "triangle" conditions on the number of completed triangles, or clustering (look at picture and try to count them, if you are curious).

Because this is a stochastic model, our outputs may differ. In order to start with the same random number, set the seed for the random number generator:

```
set.seed(0) # so we all start at the same place
flomodel.02 <- ergm(flomarriage ~ edges + triangle)
```

```
## Starting maximum pseudolikelihood estimation (MPLE):
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Finished MPLE.
## Starting Monte Carlo maximum likelihood estimation (MCMLE):
## Iteration 1 of at most 20:
## Optimizing with step length 1.
## The log-likelihood improved by 0.005369.
## Step length converged once. Increasing MCMC sample size.
## Iteration 2 of at most 20:
## Optimizing with step length 1.
## The log-likelihood improved by 0.0005893.
## Step length converged twice. Stopping.
## Finished MCMLE.
## Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 1
## This model was fit using MCMC.  To examine model diagnostics and check
## for degeneracy, use the mcmc.diagnostics() function.
```

The statement above - "this model was fit using MCMC" means - "Markov Chain Monte Carlo" method.

```
summary(flomodel.02)
```

```
## Call:
## ergm(formula = flomarriage ~ edges + triangle)
##
## Iterations:  2 out of 20
##
## Monte Carlo MLE Results:
##          Estimate Std. Error MCMC % z value Pr(>|z|)
## edges     -1.6744     0.3518      0   -4.76   <1e-04 ***
## triangle   0.1519     0.5842      0    0.26    0.795
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 166.4  on 120  degrees of freedom
##  Residual Deviance: 108.1  on 118  degrees of freedom
##
## AIC: 112.1    BIC: 117.6    (Smaller is better.)
```

OK, how do you interpret the coefficients here? Coditional log-odds of two actors forming a tie is:

=1.680 x (change in the number of ties) x 0.17 (change in the number of triangles)

For example:

- if the tie will not add any triangles to the network, its log-odds is: -1.680
- if the tie will add one triangle to the network, its log-odds is: -1.680+0.169=-1.510
- if a tie will add two triangles to the network, its log-odds is: -1.680+0.169x2=-1.341

Assignment task

Compute corresponding probabilities for the log-odds above. Are triangles significant in our model?

A closer look at the ERGM object

Let's check the model out and see what else we can do with it.

```
class(flomodel.02) # check the class
```

```
## [1] "ergm"
```

```
names(flomodel.02) # look at the ERGM object
```

```
##  [1] "coef"          "sample"        "sample.obs"    "iterations"
##  [5] "MCMCtheta"     "loglikelihood" "gradient"      "hessian"
##  [9] "covar"         "failure"       "network"       "newnetworks"
## [13] "newnetwork"    "coef.init"     "est.cov"       "coef.hist"
## [17] "stats.hist"    "steplen.hist"  "control"       "etamap"
## [21] "call"          "ergm_version"  "MPLE_is_MLE"   "formula"
## [25] "target.stats"  "target.esteq"  "constrained"   "constraints"
## [29] "reference"     "estimate"      "offset"        "drop"
## [33] "estimable"     "null.lik"      "mle.lik"
```

Now, we can check out what some of the things above mean:

```
flomodel.02$coef
```

```
##      edges   triangle
## -1.6743894  0.1519397
```

```
flomodel.02$formula
```

```
## flomarriage ~ edges + triangle
```

Assignment task

Explore the ERGM object on your own, testing at least 3-4 of the options you've generated with the *names* command. What have you learned?

Adding attributes

Of course, it would be no fun if we can't use node attributes to test their effect on the probability of forming a tie (are we talking about social selection or social influence here?). So let's go ahead and do that:

```
flomodel.03 <- ergm(flomarriage~edges+nodecov('wealth'))
```

```
## Starting maximum pseudolikelihood estimation (MPLE):
```

```
## Evaluating the predictor and response matrix.
```

```
## Maximizing the pseudolikelihood.
```

```
## Finished MPLE.

## Stopping at the initial estimate.

## Evaluating log-likelihood at the estimate.
```

```
summary(flomodel.03)
```

```
## Call:
## ergm(formula = flomarriage ~ edges + nodecov("wealth"))
##
## Iterations:  4 out of 20
##
## Monte Carlo MLE Results:
##                 Estimate Std. Error MCMC % z value Pr(>|z|)
## edges          -2.594929   0.536056      0  -4.841   <1e-04 ***
## nodecov.wealth  0.010546   0.004674      0   2.256   0.0241 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 166.4  on 120  degrees of freedom
##  Residual Deviance: 103.1  on 118  degrees of freedom
##
## AIC: 107.1    BIC: 112.7    (Smaller is better.)
```

Assignment task

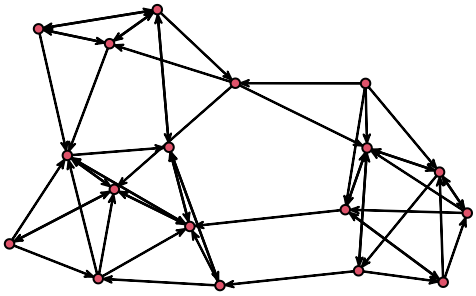
Interpet the model results. Are the coefficients significant? How does each component affect the probability of forming a tie? Calculate the corresponding probabilities.

Adding mutuality

Does the network have a tendency to create mutual ties? That is, if a tie is formed, will it be reciprocated?

We can't do this on Florentine families, but we have a different network to practice on:

```
data(samplk) # call the dataset
plot(samplk3) # take a look at the network
```



```
# Build a model and test for mutuality using the "mutual" command:
sampmodel.01 <- ergm(samplk3~edges+mutual)
```

```
## Starting maximum pseudolikelihood estimation (MPLE):
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Finished MPLE.
## Starting Monte Carlo maximum likelihood estimation (MCMLE):
## Iteration 1 of at most 20:
## Optimizing with step length 1.
## The log-likelihood improved by 0.0008421.
## Step length converged once. Increasing MCMC sample size.
## Iteration 2 of at most 20:
## Optimizing with step length 1.
## The log-likelihood improved by 0.001091.
## Step length converged twice. Stopping.
## Finished MCMLE.
## Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 1
## This model was fit using MCMC.  To examine model diagnostics and check
```

```
## for degeneracy, use the mcmc.diagnostics() function.
```

```
summary(sampmodel.01)
```

```
## Call:
## ergm(formula = samplk3 ~ edges + mutual)
##
## Iterations:  2 out of 20
##
## Monte Carlo MLE Results:
##        Estimate Std. Error MCMC % z value Pr(>|z|)
## edges   -2.1549     0.2159      0   -9.98   <1e-04 ***
## mutual   2.2990     0.4780      0    4.81   <1e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 424.2  on 306  degrees of freedom
##  Residual Deviance: 267.9  on 304  degrees of freedom
##
## AIC: 271.9    BIC: 279.3    (Smaller is better.)
```

Assignment task

Interpet the model results. Are the coefficients significant? How does each component affect the probability of forming a tie? Calculate the corresponding probabilities.

Practicing the workflow

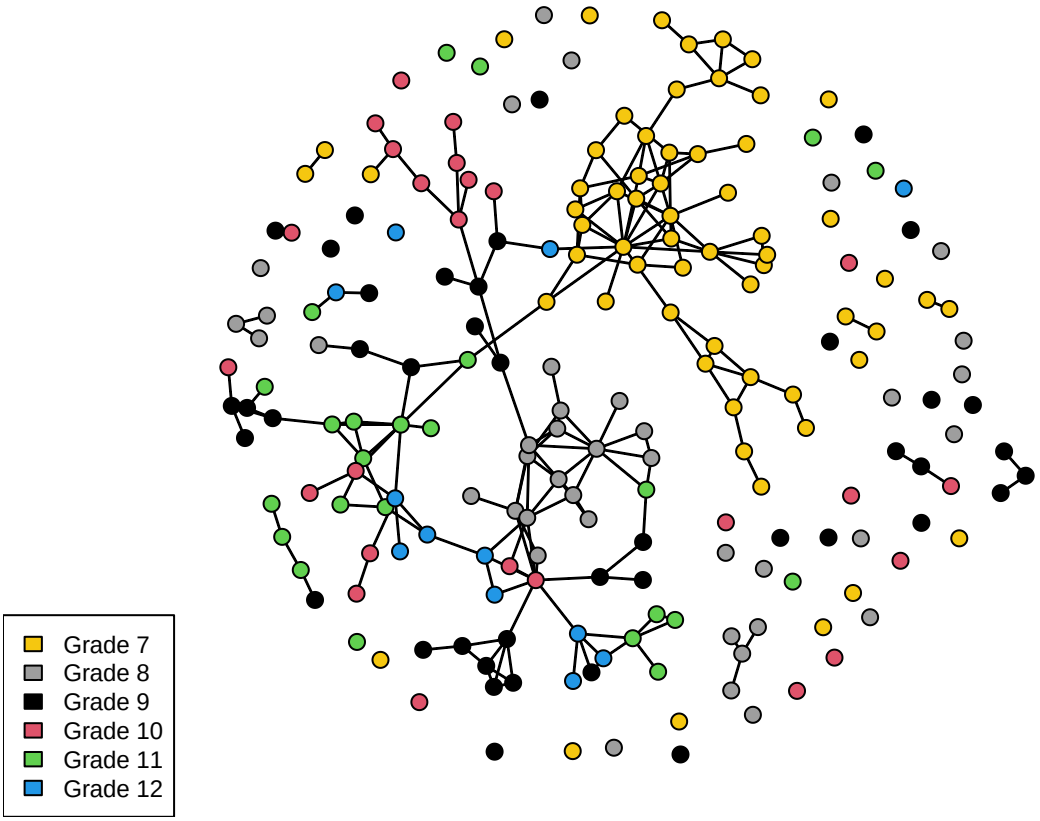
Let's take yet another dataset and run it through the few commands we have just learned, in the proper order:

1. Look at the data, plot, examine attributes. (In theory, formulate hypotheses to test.)
2. Build the unconditional model.
3. Add assortative mixing, check for model degeneracy.
4. Add linear terms for node attributes.

More complex steps we'll look at next time.

The data we get is from a high school dataset, Faux Mesa High, with students from different races, genders and grades all composing one network.

```
data(faux.mesa.high)
mesa <- faux.mesa.high
grd <- faux.mesa.high %v% "Grade"
sx <- faux.mesa.high %v% "Sex"
vs <- c(4, 12)[match(sx, c("M", "F"))]
col <- c(6, 5, 3, 7, 4, 2)
par(mar=c(0,0,0,0))
plot(mesa, vertex.col='Grade')
legend('bottomleft',fill=7:12,legend=paste('Grade',7:12),cex=0.75)
```



Let's look at our data:

```
table(mesa %v% 'Race')
```

```
##
## Black  Hisp NatAm Other White
##     6   109    68     4    18
```

```
mixingmatrix(mesa, "Race")
```

```
## Note:  Marginal totals can be misleading
##  for undirected mixing matrices.
##       Black Hisp NatAm Other White
## Black     0    8    13     0     5
## Hisp      8   53    41     1    22
## NatAm    13   41    46     0    10
## Other     0    1     0     0     0
## White     5   22    10     0     4
```

Pay attention at the table above - there are some zeros there. They will come to haunt us a little later, once we start mixing in attributes into our ERGM model.

First, the unconditional model:

```
model1 <- ergm(faux.mesa.high ~ edges)
```

```
## Starting maximum pseudolikelihood estimation (MPLE):
```

```
## Evaluating the predictor and response matrix.
```

```
## Maximizing the pseudolikelihood.
## Finished MPLE.
## Stopping at the initial estimate.
## Evaluating log-likelihood at the estimate.
```

```
summary(model1)
```

```
## Call:
## ergm(formula = faux.mesa.high ~ edges)
##
## Iterations:  7 out of 20
##
## Monte Carlo MLE Results:
##       Estimate Std. Error MCMC % z value Pr(>|z|)
## edges -4.62502    0.07053      0  -65.58   <1e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 28987  on 20910  degrees of freedom
##  Residual Deviance:  2286  on 20909  degrees of freedom
##
## AIC: 2288    BIC: 2296    (Smaller is better.)
```

Assignment task

Interpet the model results. Are the coefficients significant? How does each component affect the probability of forming a tie? Calculate the corresponding probabilities. Can you calculate the density based on the unconditional test?

Assortatative mixing

Add the assortative mixing based on grade, controlling for gender. The diff (set as true) command indicates that we make each grade different, in essence, creating a dummy set for grades.

```
model2 <- ergm(faux.mesa.high ~ edges + nodematch("Grade", diff = TRUE) + nodefactor("Sex"))
```

```
## Starting maximum pseudolikelihood estimation (MPLE):
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Finished MPLE.
## Stopping at the initial estimate.
## Evaluating log-likelihood at the estimate.
```

```
summary(model2)
```

```
## Call:
## ergm(formula = faux.mesa.high ~ edges + nodematch("Grade", diff = TRUE) +
##     nodefactor("Sex"))
##
## Iterations:  8 out of 20
##
## Monte Carlo MLE Results:
##                     Estimate Std. Error MCMC % z value Pr(>|z|)
```

```
## edges               -5.6784     0.1820      0 -31.192  < 1e-04 ***
## nodematch.Grade.7    2.7869     0.1981      0  14.071  < 1e-04 ***
## nodematch.Grade.8    2.9969     0.2395      0  12.511  < 1e-04 ***
## nodematch.Grade.9    2.4074     0.2643      0   9.107  < 1e-04 ***
## nodematch.Grade.10   2.6208     0.3744      0   7.000  < 1e-04 ***
## nodematch.Grade.11   3.3627     0.2971      0  11.319  < 1e-04 ***
## nodematch.Grade.12   3.6628     0.4578      0   8.001  < 1e-04 ***
## nodefactor.Sex.M    -0.3743     0.1047      0  -3.575  0.00035 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 28987  on 20910  degrees of freedom
##  Residual Deviance:  1915  on 20902  degrees of freedom
##
## AIC: 1931    BIC: 1995    (Smaller is better.)
```

Notice that all grades are highly significant, meaning the assortative mixing works. We do not need to separate the grades, then, and we can just use the grade as a more parsimonious linear term:

```
model2a <- ergm(faux.mesa.high ~ edges + nodematch("Grade") + nodefactor("Sex"))
```

```
## Starting maximum pseudolikelihood estimation (MPLE):

## Evaluating the predictor and response matrix.

## Maximizing the pseudolikelihood.

## Finished MPLE.

## Stopping at the initial estimate.

## Evaluating log-likelihood at the estimate.
```

```
summary(model2a)
```

```
## Call:
## ergm(formula = faux.mesa.high ~ edges + nodematch("Grade") +
##     nodefactor("Sex"))
##
## Iterations:  8 out of 20
##
## Monte Carlo MLE Results:
##                  Estimate Std. Error MCMC % z value Pr(>|z|)
## edges             -5.7011     0.1817      0 -31.383  < 1e-04 ***
## nodematch.Grade    2.8187     0.1774      0  15.889  < 1e-04 ***
## nodefactor.Sex.M  -0.3481     0.1023      0  -3.403 0.000666 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 28987  on 20910  degrees of freedom
##  Residual Deviance:  1928  on 20907  degrees of freedom
##
## AIC: 1934    BIC: 1958    (Smaller is better.)
```

Notice that the BIC term for model2a is somewhat lower than for model3 - by eliminating redundant grades, all significant, we are building a better model.

Assignment task

- Build model3, now using the nodefactor("Race"), first differentiating races, and then not, as we did with sex above. Which model is better?
- Interpet the model results. Are the coefficients significant? How does each component affect the probability of forming a tie? Calculate the corresponding probabilities.

ERGM models from start to finish

ERGM models are built in series of steps. It's not enough to just *create* a model; you have to make sure it "fits." So normally, the model creation follows several steps:

1. ERGM creation
2. MCMC diagnostics
3. ERGM model simulation
4. Model comparison
5. ERGM performance improvements

ERGM creation

Please load the *ergm* package and data from the *NetData* package.

Cleaning the data

```
library(ergm)
data(studentnets.ergm173, package = "NetData")
```

The dataset that loads contains two files - called "edges" and "nodes," where "nodes"" is a part of a larger "edges" dataset. The "edges" dataset contains information about ego's connections in semester 1 and semester 2 that we will need to pull out. The "nodes" dataset contains IDs and attributes. There are 22 nodes in total:

```
head(edges)
```

```
##   ego_id alter_id sem1_friend sem2_friend sem1_wtd_dicht_seat
## 1 104456   104456           0           1                   0
## 2 104456   114992           0           1                   0
## 3 104456   118680           0           1                   0
## 4 104456   122713           0           1                   0
## 5 104456   122723           0           1                   1
## 6 104456   125522           1           1                   0
```

```
dim(edges)
```

```
## [1] 160   5
```

```
head(nodes)
```

```
##   std_id gnd grd rce per_cap_inc
## 1 104456   2  10   4        4342
## 2 113211   2  10   1       13452
## 3 114144   1  10   4       13799
## 4 114992   1  10   4       13138
## 5 118466   1  10   2        8387
## 6 118680   2  10   4        9392
```

```
dim(nodes)
```

```
## [1] 22  5
```

The IDs in the data correspond to IDs in the complete dataset. To execute the ERGM, R requires continuous integer IDs: [1:n], where n is the total number of nodes in the ERGM. So, create node IDs acceptable to R and map these to the edges.

```
id <- seq(1,22,1)
nodes<-cbind(id, nodes)
head(nodes)
```

```
##   id std_id gnd grd rce per_cap_inc
## 1  1 104456   2  10   4        4342
## 2  2 113211   2  10   1       13452
## 3  3 114144   1  10   4       13799
## 4  4 114992   1  10   4       13138
## 5  5 118466   1  10   2        8387
## 6  6 118680   2  10   4        9392
```

Notice that we now have integer-increasing IDs as a vector in our data frame.

Next, we need to merge the new IDs from "nodes"" with the *ego_id* and *alter_id* values in edges. Between merge steps, rename variables to maintain consistency. Note that you should check each data step using earlier syntax. R requires the same ordering for node and edge-level data by ego_id, so the following sequence preserves the edgelist ordering, rendering it consistent with thenode ordering.

```
edges2<-merge(nodes[,1:2], edges, by.x = "std_id", by.y="alter_id")
head(edges2) #check the new edge dataset
```

```
##   std_id id ego_id sem1_friend sem2_friend sem1_wtd_dicht_seat
## 1 104456  1 132942           0           1                   1
## 2 104456  1 114992           1           1                   0
## 3 104456  1 126784           0           1                   0
## 4 104456  1 104456           0           1                   0
## 5 104456  1 139596           1           1                   0
## 6 113211  2 132942           1           1                   0
```

```
dim(edges2)
```

```
## [1] 144   6
```

Note that we lose some observations here. This is because the alter_id values do not exist in the node list. Search will indicate that these IDs are also not in the set of ego_id values. Let's assign new names to columns and create the final dataset:

```
names(edges2)[1]<-"alter_id"
names(edges2)[2]<-"alter_R_id"
edges3<- merge(nodes[,1:2], edges2, by.x = "std_id", by.y="ego_id") #data with IDs we need

names(edges3)[1]<-"ego_id"
names(edges3)[2]<-"ego_R_id"
head(edges3)
```

```
##   ego_id ego_R_id alter_id alter_R_id sem1_friend sem2_friend
## 1 104456        1   125522         10           1           1
## 2 104456        1   126784         12           0           1
## 3 104456        1   139596         19           1           1
## 4 104456        1   104456          1           0           1
## 5 104456        1   122713          7           0           1
## 6 104456        1   126101         11           0           1
```

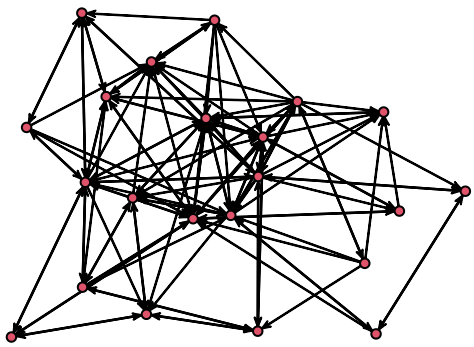
```
##   sem1_wtd_dicht_seat
## 1                   0
## 2                   0
## 3                   0
## 4                   0
## 5                   0
## 6                   0
```

The edges3 dataset now contains integer-increasing IDs sorted by ego_R_id. For our work, we will use the ego_R_id and alter_R_id values, but we retain the std_id values for reference.

Creating the network and moving onto ERGM

First, we need our network, and we'll pass some of the node attributes. You already know how to do all of that.

```
net<-network(edges3[,c("ego_R_id", "alter_R_id")]) #Dyads will be the unit of analysis
# Assign edge-level attributes - dyad attributes
set.edge.attribute(net, "ego_R_id", edges[,2])
set.edge.attribute(net, "alter_R_id", edges[,4])

# Assign node-level attributes to actors in "net"
net %v% "gender" <- nodes[,3]
net %v% "grade" <- nodes[,4]
net %v% "race" <- nodes[,5]
net %v% "pci" <- nodes[,6]
plot(net) # Look at the network
```



Now, off to the ERGM model, and you already know how to create it. Here, we are moving onto more complex model, in which we attempt to explain semester 2 friendship selections exclusively with node-level characteristics. Remember that the ERGM runs by an MCMC process with multiple starts, and this helps you see if the model is converging. If the estimated coefficient values were to change dramatically, it might be a sign of a poorly specified model. You should see the log-likelihood increase with each iteration. Remember also that you will see a loop warning, which you can ignore for now.

```
m1<-ergm(net ~ edges + mutual + nodematch("gender") + absdiff
         ("pci"),burnin=15000,MCMCsamplesize=30000,verbose=FALSE) # Incomes are coded as absolute diffe
```

```
## Starting maximum pseudolikelihood estimation (MPLE):
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Finished MPLE.
## Starting Monte Carlo maximum likelihood estimation (MCMLE):
## Iteration 1 of at most 20:
## Optimizing with step length 1.
## The log-likelihood improved by 0.001674.
## Step length converged once. Increasing MCMC sample size.
## Iteration 2 of at most 20:
## Optimizing with step length 1.
## The log-likelihood improved by 0.001952.
## Step length converged twice. Stopping.
## Finished MCMLE.
## Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
## This model was fit using MCMC.  To examine model diagnostics and check
## for degeneracy, use the mcmc.diagnostics() function.
```

Let's assess the model:

```
summary(m1)
```

```
## Call:
## ergm(formula = net ~ edges + mutual + nodematch("gender") + absdiff("pci"),
##     verbose = FALSE, burnin = 15000, MCMCsamplesize = 30000)
##
## Iterations:  2 out of 20
##
## Monte Carlo MLE Results:
##                    Estimate Std. Error MCMC % z value Pr(>|z|)
## edges            -2.314e+00  8.758e-02      0 -26.418  < 1e-04 ***
## mutual            2.402e+00  3.704e-02      1  64.858  < 1e-04 ***
## nodematch.gender  1.252e-02  8.485e-02      0   0.148 0.882724
## absdiff.pci       1.125e-04  2.893e-05      0   3.888 0.000101 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 640.5  on 462  degrees of freedom
##  Residual Deviance: 468.3  on 458  degrees of freedom
```

```
##
## AIC: 476.3    BIC: 492.9    (Smaller is better.)
```

You already know how to interpret p-star models, but let's do it again together. Before I do that, though, I'll show you another trick:

```
lapply(m1[1],exp) #this gets us a vector of exponents
```

```
## $coef
##            edges           mutual nodematch.gender      absdiff.pci
##       0.09890324      11.04685495       1.01259503       1.00011246
```

1. The first section gives the model (formula) estimated in the ERGM. Here we said that the network was a function of edges, mutual ties and matching with respect gender. Another way to think about this is that we're generating random networks that match the observed network with respect to the number of edges, the number of mutual dyads, the number of ties within between race and within/between gender. We are also considering an attribute variable, absolute value difference in incomes.
2. The second section tells how many iterations were done. The MCMC sample size is the number of random networks generated in the production of the estimate.
3. The next section gives the coefficients, their SEs and p-Values. These are on a log-odds scale, so we interpret them like logit coefficients.
4. To interpret the logit coefficients, we can take exponents - which is what we've done above. So, an edges value of -2.314 means that the odds of seeing a tie on any dyad are exp(-2.314) ~= 0.098, which could be thought of as density net of the other factors in the model. Remember, if you only have 'edges' in the model, then exp(b1) should be very close to the observed density. Moreover, a positive value of the coefficient means we have MORE edges than in a random model; negative - FEWER than in the random model. Edges are equivalent to a model intercept – while possible, I can't imagine why one would estimate a model without an edges parameter.
5. A mutual value of 2.412 means that reciprocity is more common than expected by chance (positive and significant), and here we see that exp(2.412)=11.15, so it's much more likely than chance – we are 11 times as likely to see a tie from i to j if j nominated i than if j did not nominate i.
6. We are exp(1.574e-02)=1.015 times more likely to nominate within gender than across gender. **Is this coefficient significant?**

Assignment task

1. For the model above, explain the effect of absolute differences in *pci* on tie formation.
2. For the coefficients above, calculate the associated probabilities.
3. Re-run the model with only the significant elements present. For the obtained coefficients, calculate the associated probabilities. What do you observe?

Model Diagnostics: troubleshooting and checking for model degeneracy

The final section refers to overall model fit and MCMC diagnostic statistics (AIC, BIC). But we are now ready to explore diagnostics in much more detail.

Simulation

Once we have estimated the coefficiens of an ERGM (and retained only those that we need), the model is completely specified. It defines a probability distribution across all networks of this size. If the model is a good fit to the observed data, then networks drawn from this distribution will be more likely to "resemble" the observed data. To see examples of networks drawn from this distribution we use the *simulate* command:

```
m1.sim<-simulate(m1, nsim=10) # nsim is the number of simulated models
class(m1.sim) #check what we get
```

```
## [1] "network.list"
```

```
summary(m1.sim)
```

```
## Number of Networks: 10
## Model: net ~ edges + mutual + nodematch("gender") + absdiff("pci")
## Reference: ~Bernoulli
## Constraints: ~.
## Parameters:
##            edges           mutual nodematch.gender      absdiff.pci
##    -2.3136132467     2.4021457676     0.0125163730     0.0001124573
##
## Stored network statistics:
##       edges mutual nodematch.gender absdiff.pci
##  [1,]   134     49               71      590014
##  [2,]   129     41               60      623793
##  [3,]   111     30               49      530686
##  [4,]   109     32               55      562229
##  [5,]   124     35               57      600975
##  [6,]   118     37               49      555684
##  [7,]   139     48               66      656119
##  [8,]   130     37               57      593110
##  [9,]   132     44               63      718017
## [10,]   130     43               72      657377
## attr(,"monitored")
## [1] FALSE FALSE FALSE FALSE
```

```
## Number of Networks: 10
## Model: net ~ edges + mutual + nodematch("gender") + absdiff("pci")
## Reference: ~Bernoulli
## Constraints: ~.
## Parameters:
##            edges           mutual nodematch.gender      absdiff.pci
##    -2.3136132467     2.4021457676     0.0125163730     0.0001124573
```

```
# Let's look at a couple of networks:

m1.sim[[1]] #first simulated model
```

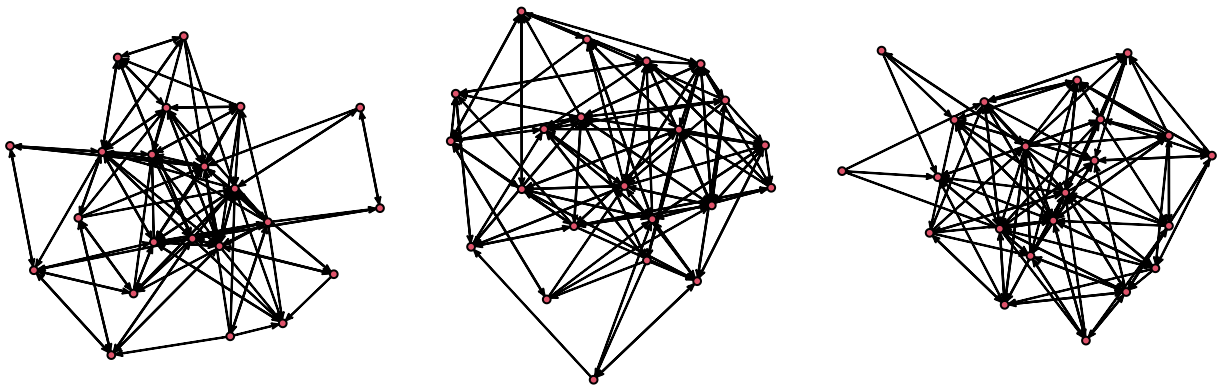
```
##  Network attributes:
##   vertices = 22
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 134
##     missing edges= 0
##     non-missing edges= 134
##
##  Vertex attribute names:
##     gender grade pci race vertex.names
##
## No edge attributes
```

```
m1.sim[[2]] #second simulated model
```

```
##  Network attributes:
```

```
##   vertices = 22
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 129
##     missing edges= 0
##     non-missing edges= 129
##
##  Vertex attribute names:
##     gender grade pci race vertex.names
##
## No edge attributes
```

```
# We can also look at them:
par(mfrow=c(1,3), mar=c(0,0,0,0))
plot(net) # original model
plot(m1.sim[[1]])
plot(m1.sim[[2]])
```



Of course, our simulated models look somewhat different from the original model. But we can use simulated results to check how good our model is.

Assignment task

For the model you created above (with only the significant coefficients) create a set of simulated models and plot a few against the original. What do you observe?

Goodness of fit, "gof"

ERGMs are generative models - that is, they represent the process that governs tie formation at a local level. These local processes in turn aggreggate to produce characteristic global network properties, even though these global properties are not explicit in the model. One test of whether a model "fits the data" is how well it reproduces these global properties. We do this by choosing a network statistic that is not in the model and comparing the value of this statistic observed in the original network to the distribution of values we get in simulated networks from our model.

Such network statistics can be many, for example: indegree, outdegree, transitivity, etc. However, in the *ergm* package, they are named differently. Below is a list of a few, but please consult the help manual to find the rest when you need them.

Some ERGM terms for comparing statistics:

- distance (geodesic distance)
- espartners (shared partner distributions, edgewise)
- dspartners
- odegree
- idegree
- degree (for undirected network)
- triadcensus
- model (terms of the original model)

So let's go ahead and look at our simulated model with indegree (since it's a valued network) as a network statistic we compare:

```
library(knitr)
```

```
## Warning: package 'knitr' was built under R version 4.0.5
```

```
m1.gof<-gof(m1~idegree)
# The code below is commented out, but if you want to see
# what we get with m1.gof object, check it:
##names(m1.gof)
kable(m1.gof$pval.ideg, caption="Goodness-of-fit for Indegree")
```

Table 1: Goodness-of-fit for Indegree

	obs	min	mean	max	MC p-value
0	0	0	0.06	1	1.00
1	1	0	0.32	3	0.54
2	2	0	1.06	4	0.52
3	5	0	2.07	6	0.04
4	1	0	3.28	7	0.26
5	3	0	3.74	9	0.96
6	2	0	3.90	11	0.40
7	2	0	3.18	8	0.74
8	0	0	1.87	6	0.38

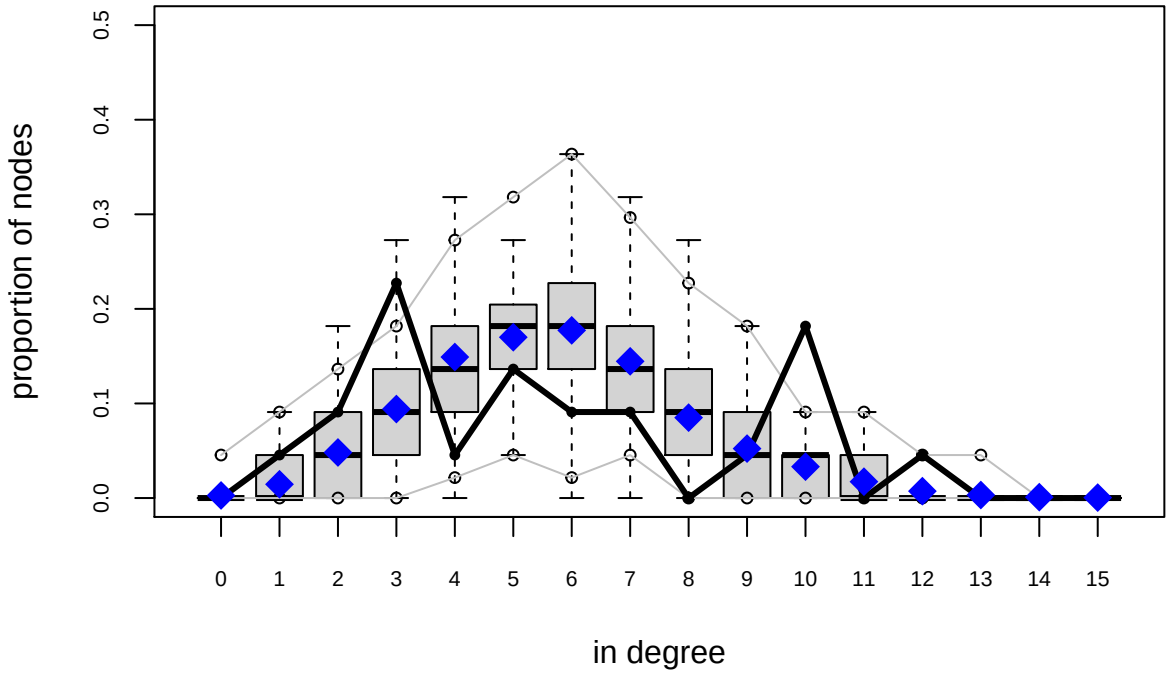
	obs	min	mean	max	MC p-value
9	1	0	1.15	4	1.00
10	4	0	0.73	4	0.02
11	0	0	0.38	3	1.00
12	1	0	0.16	1	0.32
13	0	0	0.07	1	1.00
14	0	0	0.02	1	1.00
15	0	0	0.01	1	1.00
16	0	0	0.00	0	1.00
17	0	0	0.00	0	1.00
18	0	0	0.00	0	1.00
19	0	0	0.00	0	1.00
20	0	0	0.00	0	1.00
21	0	0	0.00	0	1.00

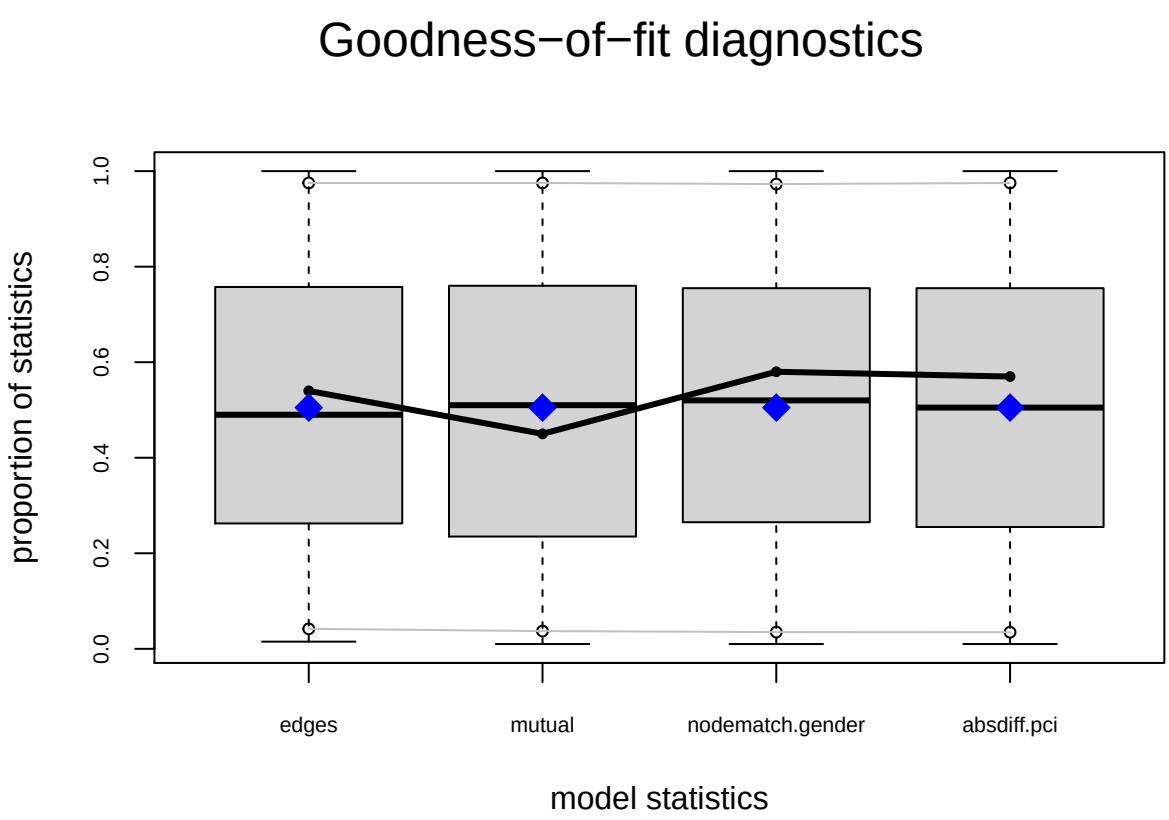
The table is somewhat not intuitive, because the first column is actually not a row number, but the indegree that a node can have. In our network, we simulated a possibility for the indegree of 0 (no ties directed to the node) to 21 (maximum of 21 ties to any one node from the other 21 nodes for the total of 22 nodes in the network).

What does it mean to have a significant result? We are used to have p-value be less than 0.05 for a good model, where we REJECT the null hypotheses of NO relationship. With similation modeling, when we are evaluating the model fit, we are looking for the opposite, actually - where we CANNOT reject the null hypotheses of NO difference. If p-value is significant, that means our model and the simulated model ARE different, so the fit is NOT good.

Let's look at the same thing graphically:

```
par(mfrow=c(1,1))
plot(m1.gof)
```


proportion of nodes
0.0
0.1
0.2
0.3
0.4
0.5
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
in degree



GOF figure: the solid line in each plot represents the observed statistics for our m1. network, and the boxplots summarize the statistics for the simulated networks resulting from the MLE. Pay attention to the axis labels! They represent degree (in our case, max is 15, because at indegree=14 we have the last mean value greater than 0) on the x-axis and the proportion of nodes with that indegree on the y-axis.

Assignment task

For the model you created above (with only the significant coefficients) run a goodness-of-fit test with any network characteristic you choose, which is not already in the model. Generate a table with information and a plot. What do you observe?

Markov Chain Monte Carlo, MCMC diagnostics

The computational algorithm in *ergm* uses MCMC to estimate the likelihood function. Part of this process involves simulating a set of networks to approximate unknown components of the likelihood.

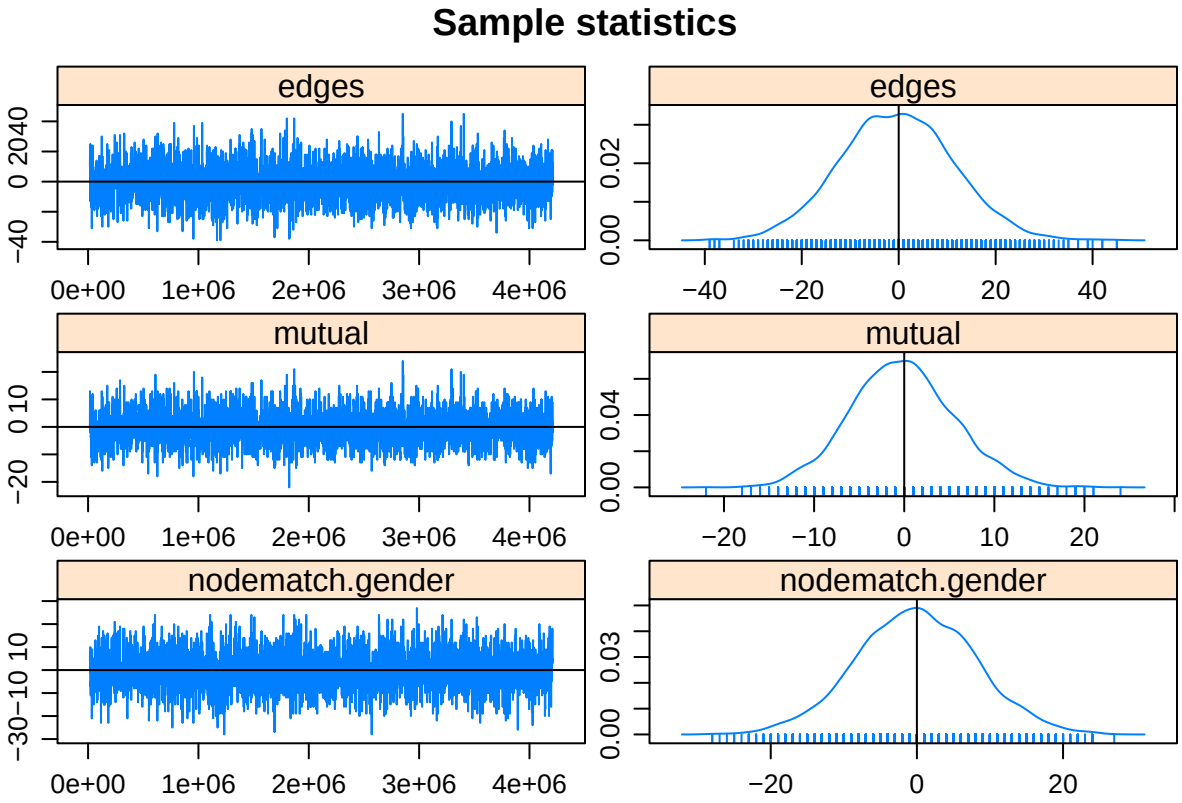
When a model is not a good representation of the observed network the estimation process may be affected. In the worst case scenario, the simulated networks will be so different from the observed network that the algorithm fails altogether. This can occur for two general reasons. First, the sim-ulation algorithm may fail to converge, and the sampled networks are thus not from the specified distribution. Second, the model parameters used to simulate the networks are too different from the MLE, so even though the simulation algorithm is producing a representative sample of networks, this is not the sample that would be produced under the MLE (for mode details, see the papers I've given you in the last seminar).

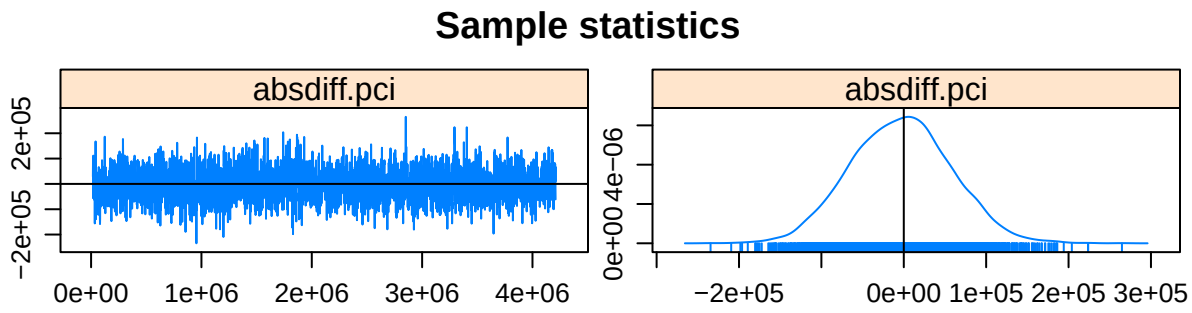
We can use diagnostics to see what is happening with the simulation algorithm, and these can lead us to ways to improve it.

```
par(mar=c(0,0,0,0))
mcmc.diagnostics(m1)
```

```
## Sample statistics summary:
##
## Iterations = 16384:4209664
## Thinning interval = 1024
## Number of chains = 1
## Sample size per chain = 4096
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                        Mean        SD  Naive SE Time-series SE
## edges               -0.3779    11.696   0.18274         0.2312
## mutual              -0.2058     5.742   0.08972         0.1197
## nodematch.gender    -0.1221     8.129   0.12701         0.1648
## absdiff.pci      -3331.4238 60899.626 951.55666      1278.4563
##
## 2. Quantiles for each variable:
##
##                     2.5%    25%   50%   75%  97.5%
## edges                -23     -8     0     7     22
## mutual               -11     -4     0     3     11
## nodematch.gender     -16     -6     0     5     16
## absdiff.pci      -119279 -45687 -2822 37304 115507
##
##
## Sample statistics cross-correlations:
##                        edges    mutual nodematch.gender absdiff.pci
## edges              1.0000000 0.8714879        0.6978460   0.8447108
## mutual             0.8714879 1.0000000        0.6135576   0.7799928
## nodematch.gender   0.6978460 0.6135576        1.0000000   0.6120116
## absdiff.pci        0.8447108 0.7799928        0.6120116   1.0000000
##
## Sample statistics auto-correlation:
## Chain 1
##                edges     mutual nodematch.gender absdiff.pci
## Lag 0    1.000000000 1.00000000       1.00000000 1.000000000
## Lag 1024 0.230658335 0.25885464       0.25463122 0.286902166
## Lag 2048 0.066768882 0.08849163       0.07054685 0.102318194
## Lag 3072 0.023843175 0.01730998       0.04767334 0.036800373
## Lag 4096 0.005882174 0.01395031       0.03086641 0.013640076
## Lag 5120 0.010224870 0.01721133       0.03152573 0.004862375
##
## Sample statistics burn-in diagnostic (Geweke):
## Chain 1
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##            edges           mutual nodematch.gender      absdiff.pci
##           0.3192          -0.1594           0.6258           0.3848
##
```

```
## Individual P-values (lower = worse):
##             edges            mutual nodematch.gender       absdiff.pci
##         0.7495994         0.8733637         0.5314224         0.7003803
## Joint P-value (lower = worse):  0.9968117 .
```





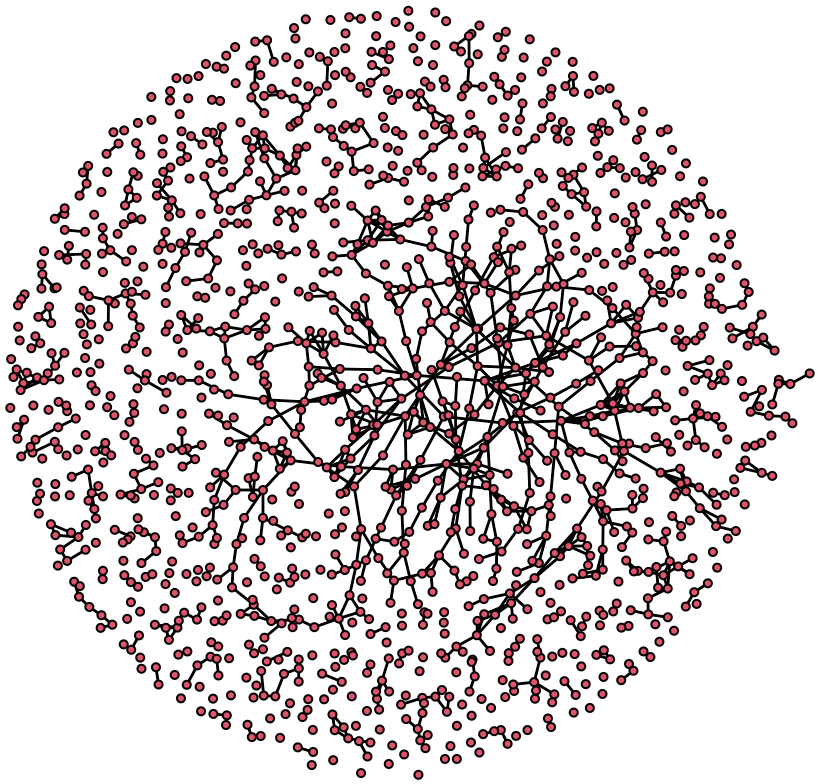

```
##
## MCMC diagnostics shown here are from the last round of simulation, prior to computation of final para
```

You see several plots and output. The plots should look approximately normal, with a peak of a distribution centered at zero. Remember, when looking at p-values, we DO NOT want significant results - we actually want to find NO DIFFERENCE between our model and the simulated model, because it means the model fits our data well.

In our case the samples might be too small. This doesn't mean the results of the ERGM results are wrong, but we should take care in specifying the sample size.

To see how this actually works on a better set of data, let's consider another dataset. Here, we are evaluating the ERGM model on edges and triangles, and to get the same results, I am starting on seed(1).

```
data('faux.magnolia.high')
magnolia <- faux.magnolia.high
par(mar=c(0,0,0,0))
plot(magnolia, vertex.cex=.5)
```



Next, we need to fit the ERGM model, but the Rmarkdown chokes on the result. Therefore, I copy for you the output, but comment out the code. You can try it on your own:

```
#fit <- ergm(magnolia~edges+triangle, control=control.ergm(seed=1234))
```

Error in ergm.MCMLE(init, nw, model, initialfit = (initialfit <- NULL),:

Number of edges in a simulated network exceeds that in the observed by a factor of more than 20. This is a strong indicator of model degeneracy or a very poor starting parameter configuration. If you are reasonably certain that neither of these is the case, increase the MCMLE.density.guard control.ergm() parameter.

Very interesting. In the process of trying to fit this model, the algorithm heads off into networks that are much much more dense than the observed network. This is a big problem, and such a clear indicator of degeneracy that the algorithm cuts off to avoid heading off into areas that would cause memory issues. If you'd like to peak a bit more under the hood, you can stop the algorithm in time to catch where it's heading (notice I am using the control.ergm function to do that; read up on it to see what it's doing):

```
fit <- ergm(magnolia~edges+triangle, control=control.ergm(seed=0,MCMLE.maxit=1, MCMLE.density.guard.min
```

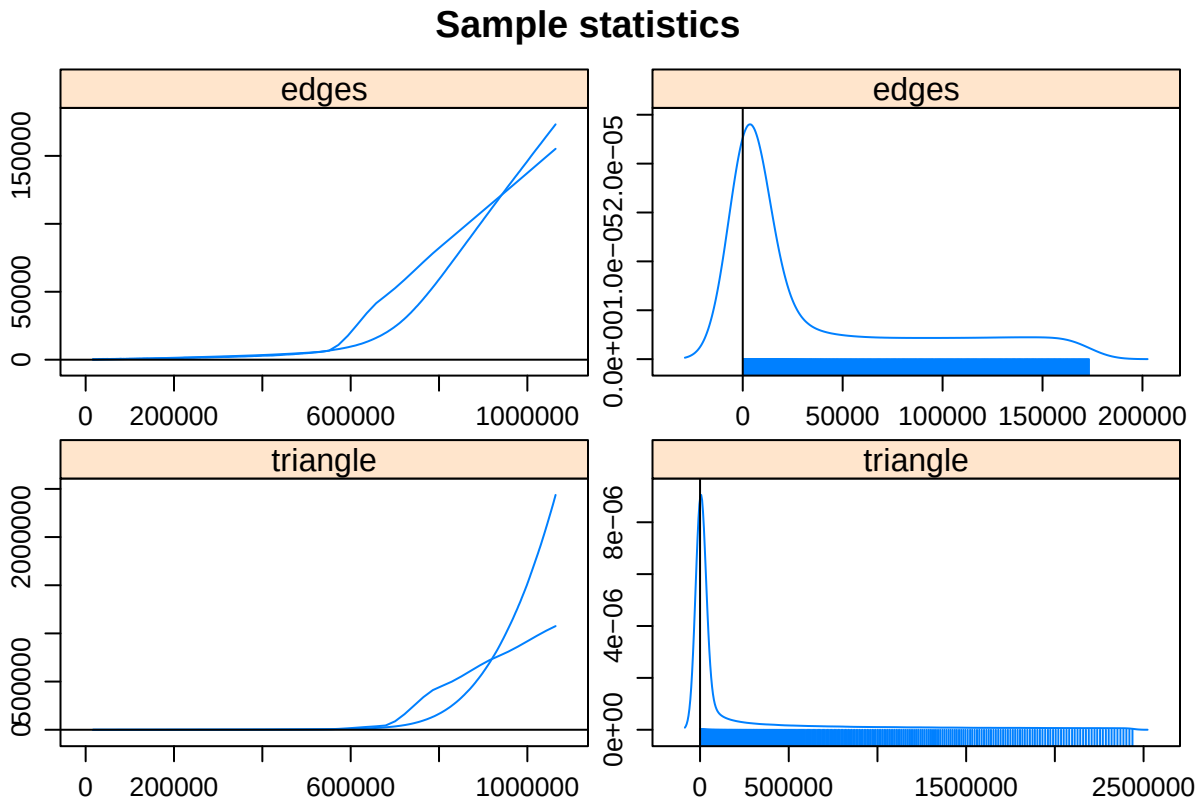
```
## Starting maximum pseudolikelihood estimation (MPLE):

## Evaluating the predictor and response matrix.

## Maximizing the pseudolikelihood.

## Finished MPLE.

## Starting Monte Carlo maximum likelihood estimation (MCMLE):

## Iteration 1 of at most 1:

## Optimizing with step length 0.930409958187668.
```

```
## The log-likelihood improved by 0.3188.
## MCMLE estimation did not converge after 1 iterations. The estimated coefficients may not be accurate
## Finished MCMLE.
## Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 1
## This model was fit using MCMC.  To examine model diagnostics and check
## for degeneracy, use the mcmc.diagnostics() function.
```

```
mcmc.diagnostics(fit)
```

```
## Sample statistics summary:
##
## Iterations = 16384:1063936
## Thinning interval = 1024
## Number of chains = 1
## Sample size per chain = 1024
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean     SD Naive SE Time-series SE
## edges     36267  50943     1592          36570
## triangle 260603 538847    16839         263669
##
## 2. Quantiles for each variable:
##
##            2.5%    25%  50%    75%   97.5%
## edges    523.73 1711.0 6194  59983  162189
## triangle  53.58  774.2 5016 170445 2026202
##
##
## Sample statistics cross-correlations:
##              edges  triangle
## edges    1.0000000 0.9288222
## triangle 0.9288222 1.0000000
##
## Sample statistics auto-correlation:
## Chain 1
##              edges  triangle
## Lag 0    1.0000000 1.0000000
## Lag 1024 0.9962134 0.9918680
## Lag 2048 0.9924241 0.9837793
## Lag 3072 0.9886326 0.9757324
## Lag 4096 0.9848373 0.9677247
## Lag 5120 0.9810400 0.9597589
##
## Sample statistics burn-in diagnostic (Geweke):
## Chain 1
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##    edges triangle
##   -1.484   -1.162
```

```
##
## Individual P-values (lower = worse):
##     edges  triangle
## 0.1377052 0.2452073
## Joint P-value (lower = worse):  0.5574559 .
```




```
##
## MCMC diagnostics shown here are from the last round of simulation, prior to computation of final para
```

Look at the graphics! Clearly, it's heading somewhere very bad. Our triangles and edges are increasing almost exponentially, creating an infinitely dense network.

This is not the first time such problem appeared. To combat it, a more robust algorithm of estimating edges, GWESP, was developed.

```
fit <- ergm(magnolia~edges+gwesp(0.5,fixed=T),control = control.ergm(seed=1))
```

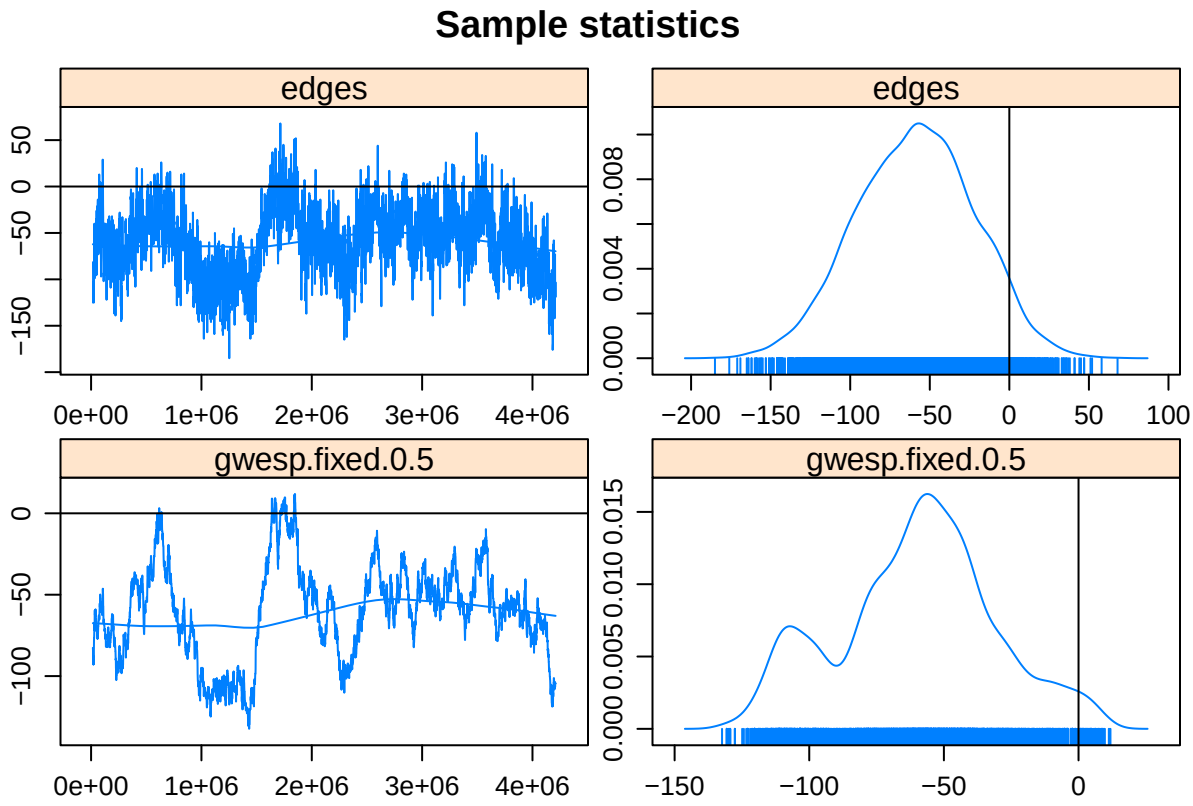
```
## Starting maximum pseudolikelihood estimation (MPLE):

## Evaluating the predictor and response matrix.

## Maximizing the pseudolikelihood.

## Warning in ergm.mple(nw, fd, m, MPLEtype = MPLEtype, init = init, control = control, : GLM model may

## Finished MPLE.

## Starting Monte Carlo maximum likelihood estimation (MCMLE):

## Iteration 1 of at most 20:

## Optimizing with step length 1.
```

```
## The log-likelihood improved by 2.693.
## Step length converged once. Increasing MCMC sample size.
## Iteration 2 of at most 20:
## Optimizing with step length 0.53527301111519.
## The log-likelihood improved by 2.895.
## Iteration 3 of at most 20:
## Optimizing with step length 1.
## The log-likelihood improved by 0.001371.
## Step length converged once. Increasing MCMC sample size.
## Iteration 4 of at most 20:
## Optimizing with step length 1.
## The log-likelihood improved by 2.133.
## Step length converged twice. Stopping.
## Finished MCMLE.
## Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 1
## This model was fit using MCMC.  To examine model diagnostics and check
## for degeneracy, use the mcmc.diagnostics() function.
```

```
mcmc.diagnostics(fit)
```

```
## Sample statistics summary:
##
## Iterations = 16384:4209664
## Thinning interval = 1024
## Number of chains = 1
## Sample size per chain = 4096
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                    Mean    SD Naive SE Time-series SE
## edges           -58.83 36.93   0.5770          6.637
## gwesp.fixed.0.5 -60.44 29.29   0.4577          9.503
##
## 2. Quantiles for each variable:
##
##                  2.5%    25%    50%    75%  97.5%
## edges            -131 -85.00 -58.00 -33.00 10.000
## gwesp.fixed.0.5  -115 -78.33 -58.22 -42.18 -1.188
##
##
## Sample statistics cross-correlations:
##                      edges gwesp.fixed.0.5
## edges           1.0000000       0.7435124
## gwesp.fixed.0.5 0.7435124       1.0000000
##
## Sample statistics auto-correlation:
## Chain 1
```

```
##              edges gwesp.fixed.0.5
## Lag 0    1.0000000       1.0000000
## Lag 1024 0.8183931       0.9951459
## Lag 2048 0.7114994       0.9905437
## Lag 3072 0.6464956       0.9860104
## Lag 4096 0.6063466       0.9812088
## Lag 5120 0.5810402       0.9764216
##
## Sample statistics burn-in diagnostic (Geweke):
## Chain 1
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##           edges gwesp.fixed.0.5
##          -1.215          -1.270
##
## Individual P-values (lower = worse):
##           edges gwesp.fixed.0.5
##       0.2244260       0.2042196
## Joint P-value (lower = worse):  0.1906356 .
```




```
##
## MCMC diagnostics shown here are from the last round of simulation, prior to computation of final para
```

A little better, though there are still issues. Let's try tweaking the GWESP model a bit:

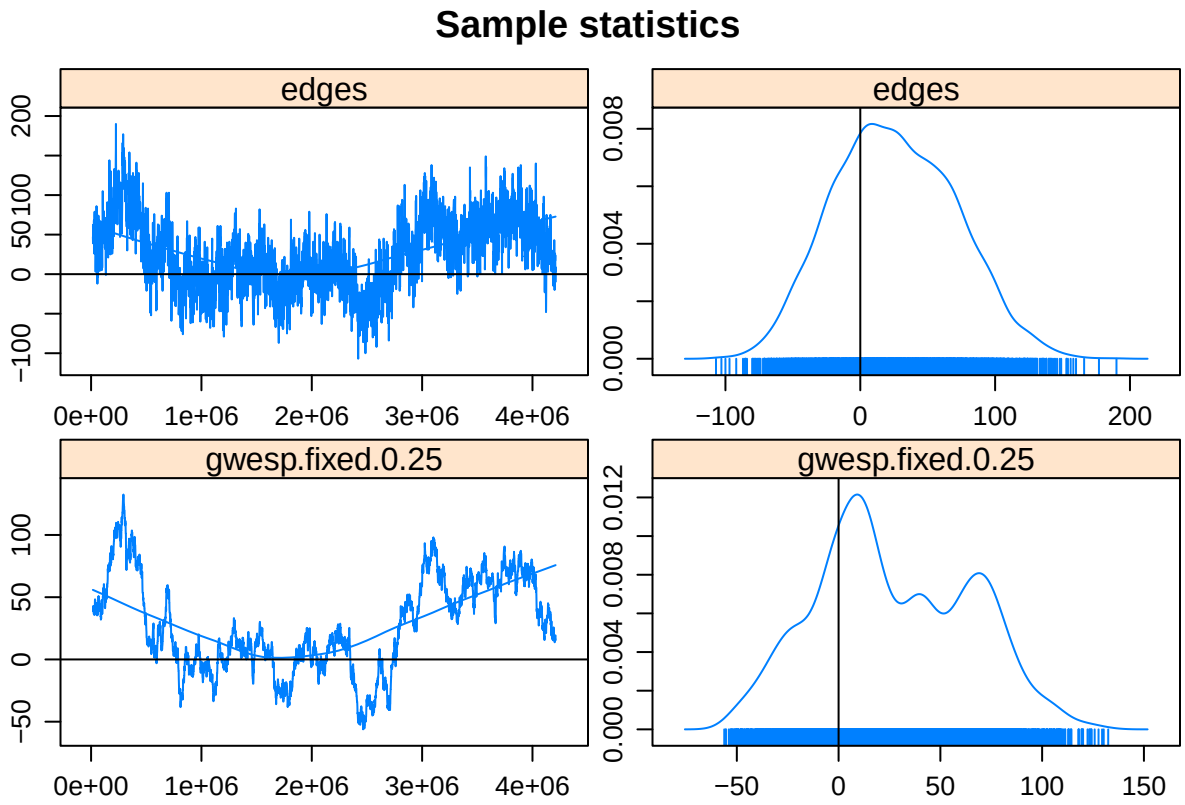
```
fit <- ergm(magnolia~edges+gwesp(0.25,fixed=T),control = control.ergm(seed=1))
```

```
## Starting maximum pseudolikelihood estimation (MPLE):
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Warning in ergm.mple(nw, fd, m, MPLEtype = MPLEtype, init = init, control = control, : GLM model may
## Finished MPLE.
## Starting Monte Carlo maximum likelihood estimation (MCMLE):
## Iteration 1 of at most 20:
## Optimizing with step length 1.
## The log-likelihood improved by 4.505.
## Step length converged once. Increasing MCMC sample size.
## Iteration 2 of at most 20:
## Optimizing with step length 1.
## The log-likelihood improved by 0.2551.
## Step length converged twice. Stopping.
## Finished MCMLE.
## Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
## This model was fit using MCMC.  To examine model diagnostics and check
## for degeneracy, use the mcmc.diagnostics() function.
```

```
mcmc.diagnostics(fit)
```

```
## Sample statistics summary:
##
## Iterations = 16384:4209664
## Thinning interval = 1024
## Number of chains = 1
## Sample size per chain = 4096
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                    Mean    SD Naive SE Time-series SE
## edges             25.77 45.11   0.7049          11.55
## gwesp.fixed.0.25 26.96 37.82   0.5910          15.67
##
## 2. Quantiles for each variable:
##
##                     2.5%     25%   50%   75%  97.5%
## edges             -55.00 -7.0000 24.00 58.00 116.00
## gwesp.fixed.0.25 -39.17 -0.4583 19.91 59.71  97.97
##
##
## Sample statistics cross-correlations:
##                       edges gwesp.fixed.0.25
## edges             1.0000000        0.8372046
```

```
## gwesp.fixed.0.25 0.8372046        1.0000000
##
## Sample statistics auto-correlation:
## Chain 1
##              edges gwesp.fixed.0.25
## Lag 0    1.0000000        1.0000000
## Lag 1024 0.8781759        0.9971581
## Lag 2048 0.8053131        0.9942111
## Lag 3072 0.7638662        0.9912357
## Lag 4096 0.7344345        0.9883461
## Lag 5120 0.7152067        0.9853563
##
## Sample statistics burn-in diagnostic (Geweke):
## Chain 1
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##            edges gwesp.fixed.0.25
##            3.214            1.568
##
## Individual P-values (lower = worse):
##            edges gwesp.fixed.0.25
##      0.001308153      0.116958890
## Joint P-value (lower = worse):  0.4701631 .
```




```
##
```

```
## MCMC diagnostics shown here are from the last round of simulation, prior to computation of final para
```

One more try. Here we'll change two things - add some more reasonable terms to the model, and add in some robustness to the fitting algorithm by making the MCMC simulation longer. In practice it might take more trial and error to find this solution:

```
fit <- ergm(magnolia~edges+gwesp(0.25,fixed=T)+
            nodematch('Grade')+nodematch('Race')+nodematch('Sex'),
            control = control.ergm(seed=1,MCMC.samplesize=50000,MCMC.interval=1000), verbose=T)
```

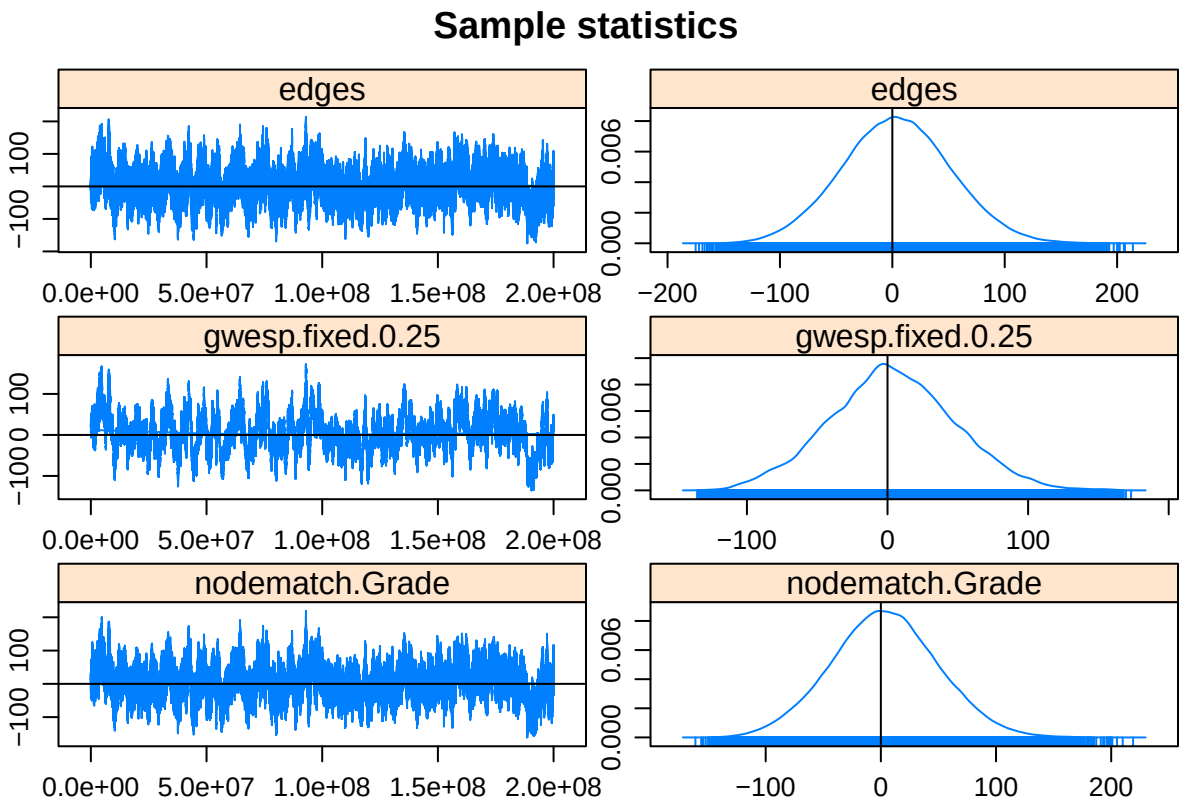
```
## Evaluating network in model.
## Initializing Metropolis-Hastings proposal(s): ergm:MH_TNT
## Initializing model.
## Using initial method 'MPLE'.
## Fitting initial model.
## Starting maximum pseudolikelihood estimation (MPLE):
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Finished MPLE.
## Starting Monte Carlo maximum likelihood estimation (MCMLE):
## Density guard set to 19563 from an initial count of 974 edges.
##
## Iteration 1 of at most 20 with parameter:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##       -9.6939224        1.6624034        2.8170436        0.9870588
##    nodematch.Sex
##        0.6188166
## Starting unconstrained MCMC...
## Back from unconstrained MCMC.
## Average estimating function values:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##         27.46432        -39.64924         22.73978         29.37336
##    nodematch.Sex
##        -22.90568
## Starting MCMLE Optimization...
## Optimizing with step length 0.993395864420747.
## Using lognormal metric (see control.ergm function).
## Using log-normal approx (no optim)
## The log-likelihood improved by 7.361.
##
## Iteration 2 of at most 20 with parameter:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##       -9.7737769        1.7986284        2.7564779        0.9028708
##    nodematch.Sex
##        0.7661820
## Starting unconstrained MCMC...
## Back from unconstrained MCMC.
## Average estimating function values:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##         19.16178         14.67625         18.98292         16.22280
##    nodematch.Sex
##         15.18320
## Starting MCMLE Optimization...
## Optimizing with step length 1.
## Using lognormal metric (see control.ergm function).
```

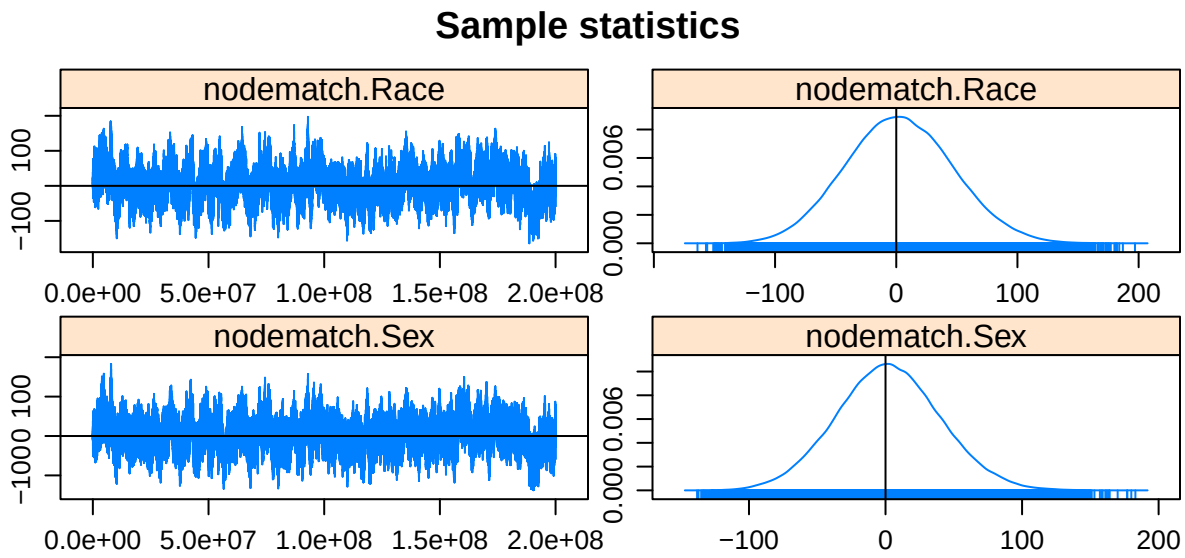
```
## Using log-normal approx (no optim)
## The log-likelihood improved by 0.1034.
## Step length converged once. Increasing MCMC sample size.
##
## Iteration 3 of at most 20 with parameter:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##       -9.7718057        1.7994968        2.7457210        0.9060500
##    nodematch.Sex
##        0.7592927
## Starting unconstrained MCMC...
## Back from unconstrained MCMC.
## Average estimating function values:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##         3.835765         3.503317         3.441665         2.788565
##    nodematch.Sex
##         2.258610
## Starting MCMLE Optimization...
## Optimizing with step length 1.
## Using lognormal metric (see control.ergm function).
## Using log-normal approx (no optim)
## Starting MCMC s.e. computation.
## The log-likelihood improved by 0.005033.
## Step length converged twice. Stopping.
## Finished MCMLE.
## Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
## This model was fit using MCMC.  To examine model diagnostics and check
## for degeneracy, use the mcmc.diagnostics() function.
```

```
mcmc.diagnostics(fit)
```

```
## Sample statistics summary:
##
## Iterations = 16000:200015000
## Thinning interval = 1000
## Number of chains = 1
## Sample size per chain = 2e+05
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                    Mean    SD Naive SE Time-series SE
## edges             3.836 48.29  0.10798          2.496
## gwesp.fixed.0.25 3.503 44.37  0.09922          3.606
## nodematch.Grade  3.442 46.36  0.10367          2.635
## nodematch.Race   2.789 44.27  0.09899          2.449
## nodematch.Sex    2.259 38.16  0.08533          1.855
##
## 2. Quantiles for each variable:
##
##                     2.5%    25%   50%   75% 97.5%
## edges             -90.00 -29.00 4.000 36.00 99.00
## gwesp.fixed.0.25 -83.39 -26.16 2.687 32.75 92.03
## nodematch.Grade  -87.00 -28.00 3.000 34.00 96.00
## nodematch.Race   -82.00 -28.00 2.000 33.00 91.00
## nodematch.Sex    -72.00 -23.00 2.000 28.00 79.00
```

```
##
##
## Sample statistics cross-correlations:
##                      edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
## edges            1.0000000       0.8617826       0.9636771      0.9465237
## gwesp.fixed.0.25 0.8617826       1.0000000       0.8816586      0.8558440
## nodematch.Grade  0.9636771       0.8816586       1.0000000      0.9200879
## nodematch.Race   0.9465237       0.8558440       0.9200879      1.0000000
## nodematch.Sex    0.9094978       0.8017419       0.8786393      0.8667274
##                  nodematch.Sex
## edges                0.9094978
## gwesp.fixed.0.25     0.8017419
## nodematch.Grade      0.8786393
## nodematch.Race       0.8667274
## nodematch.Sex        1.0000000
##
## Sample statistics auto-correlation:
## Chain 1
##              edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
## Lag 0    1.0000000       1.0000000       1.0000000      1.0000000
## Lag 1000 0.9427469       0.9978793       0.9646558      0.9549927
## Lag 2000 0.9021754       0.9958011       0.9369907      0.9220824
## Lag 3000 0.8727179       0.9937367       0.9146770      0.8970839
## Lag 4000 0.8503503       0.9917068       0.8961979      0.8775981
## Lag 5000 0.8326313       0.9897267       0.8806069      0.8619862
##          nodematch.Sex
## Lag 0        1.0000000
## Lag 1000     0.9459135
## Lag 2000     0.9067483
## Lag 3000     0.8774160
## Lag 4000     0.8541598
## Lag 5000     0.8353573
##
## Sample statistics burn-in diagnostic (Geweke):
## Chain 1
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##            2.527           1.848            2.395            2.363
##    nodematch.Sex
##            2.305
##
## Individual P-values (lower = worse):
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##       0.01149822       0.06455497       0.01662109       0.01814756
##    nodematch.Sex
##       0.02117798
## Joint P-value (lower = worse):  0.4188632 .
```


Sample statistics
edges
edges
gwesp.fixed.0.25
gwesp.fixed.0.25
nodematch.Grade
nodematch.Grade
−100 100
0.000 0.006
0.0e+00 5.0e+07 1.0e+08 1.5e+08 2.0e+08
−200 −100 0 100 200
−100 0 100




```
##
## MCMC diagnostics shown here are from the last round of simulation, prior to computation of final para
```

Success! Here, we got lucky. In real life, we might have to do a lot more tweaking in order to obtain similar results.

Homework 2

For this assignment, you'll be working with a real-life dataset collected by our lab.

The data

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.

Ground rules

This is a real-life dataset collected on a real-life company, as I've mentioned. Before we go into any further details with it, the following are the ground rules:

- This dataset is for educational purposes only, for your use in this class. You may NOT, under any circumstances, distribute it outside of this class, for any reason.
- To use this dataset for any purposes other than this course, even if it is for your diploma, you MUST obtain my permission IN WRITING. Referring later to a "conversation" we might have had about it will not suffice.

- If any paper is published from this dataset, in addition to obtaining a permission, you must include the authors of this dataset as co-authors. I will give you the names if and when such time comes.

Dataset description

The data was collected in a real-life company, and employees were asked to answer questions of several types. We were interested in their attitudes towards their jobs, as well as levels of burnout, conflict, etc., in the organization. We were also interested in whether the employees were exhibiting the so-called "organizational citizenship behavior" (please read about this concept a little as you work with this dataset), so we collected additional questions on that. To reduce common method bias, we collected data from employees themselves, and measures of their performance - from their supervisors.

There is very little missing data, so we were lucky with our response rates. All people surveyed were the people employed in one department - there were 71 people in total, we got data on 68. We are missing just three from that department, though network data is available on many more - those are people from other departments.

Other than describe the dataset for you, I am going to refer you to the file "Questionnaire.pdf" you will find question descriptions right there.

- First page is introduction and description. It is a promise to keep information confidential (and you will see the measures we've taken to protect the confidentiality) and contact information of the researcher (redacted for now).
- Second page collects demongraphic informatin of the respondent. It corresponds to the first 12 columns of the datafile "OCB," the "Main" tab. When you open the file, you will see that some questions have changed their order, and they've been named as short descriptions of the questions, but everything is there.
- Network questions. This is how we collect the multiplex data. We ask for the person's name, then ask to describe the contact type, and for each selected contact type we ask the person to value the relationship, as indicated in the questionnaire. This is the real questionnaire we've used, so network questions take a few pages.

  People were free to name as many people as they've wished and the people they named did not need to be from their own department. So we ended up with 68 responses from one department and 122 people as part of the network. On the remaining 54 we do not have demographic information, because they did not participate in our survey.
- Substantive questions. This is the part where we collect the information of interest. This survey had A LOT of questions - and they are grouped together by topic. Whether it is correct is subject of debate, but we opted to do it that way based on solid methodological recommendations, trust me.))
- Performance data. This part of the questionnaire was filled by the supervisors only. They had to list their employees, and then rate them on questions asked, which evaluated both their task performance and the OCB.

All collected questionnaires were entered into Excel for further analysis. In the file "OCB" you will find the following information:

- Tab "Main" contains all demographic information on our respondents and their answers to substantive questions (work attitudes). Please note that respondent names are reduced to their initials (in English), so that we can protect their privacy. Combined with not knowing the company or the department, this assures respondent privacy.

  Questions Q1-Q37 correspond to questions 1:1-1:29 (first 29) and 2:1-2:8 (second eight). Questions QS1-QS36 correspond to supervisors' ratings of these employees.

  Please note: Some questions will have a letter "R" to them. This is because they were REVERSE-CODED. I am sure you know what reverse-coding means, so we had to transfer data back to the same

scale as non-reverse-coded items, so that we could put them together into one factor. In this dataset, all reverse-coding was complete, so you do not need to worry about that.

- Tabs "Friendship," "Professional," "Boss" and "Support" are adjacency matrices - respondents are entered in exact the same order in rows and columns of all matrices. Data in these matrices is shown as is, without adding additional information - just the valued responses to connections that people have indicated.
- Tab "All Net" is the network of connections that's dichotomized somewhat the same way we've dichotomized the "trade.all" data - here, if any type of connection is present in either of the previous tabs, there is a "1" in the cell. Otherwise, a zero. This is already a matrix with zeros in cells that do not have a value.
- Tabs "*.Net" - adjacency data from previous tabs is converted to networks for further use.

Assignment Task 1

1. Please carefully look at the Questionnaire. For the employees, questions were broken down into groups, and groups were even named - those are factors that we can analyze using Factor Analysis. Which of these factors could be used for good dependent variables? Why?
2. Please examine the supervisor's questionnaire. In there, questions were not grouped (by the way, can you tell the reason why?). Look at questions carefully, and using face validity, try to determine which questions would form separate and theoretically justifiable factors.

Extracting data for further use

We can turn data into many different storage formats, but since we are starting with Excel, I find it the easiest to just convert data into .csv. As you may know, .csv does not allow to have tabs - so you have to extract one worksheet at a time. It's not that bad, actually, considering that we might want to store all that data separately, anyway.

Extracting data from files

All attribute data, including responses to content questions, are stored in the tab "Main," where column names are variable names, and rows contain the information. To extract the data, do the following:

- Right-mouse-click on the tab, select "Move or Copy."
- In the field "Move selected sheets to book" select "New book"
- Check-mark "Create a copy," click OK. A copy of the tab will appear as a separate file.
- Select File->Save As. Find your working directory, save the file as "OCB_att" (or any other name - give it my name if you want to copy my code), give it extension "CSV(Comma delimited)" in the drop-down menu. Click "yes" when Excel gives you a warming about losing some of your storage structure, it's OK.
- For the purpose of this seminar, I will also move the "AllNet" tab into the .csv format; you will be exploring the rest of the data on your own. The only difference with the attribute file is that you have names of variables in both rows and columns. I recommend you REMOVE the first columns with names and only leave the names in rows. It greatly simplifies working with the file, because you do not need to clean it up once you bring it in.

Loading network data

Here is how to create a network out of "AllNet" file (first, we have to create "AllNet.csv" from our main file):

```
all_net<-read.csv('AllNet.csv', header=TRUE) # read data
all_mat<-as.matrix(all_net) # save it as a matrix
```

Next, let's create a graph:

```
library(igraph)
```

```
##
## 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
```

```
all_graph<-graph_from_adjacency_matrix(all_mat) #create a graph
```

Let's load the rest of the networks:

```
FriendMat<-read.csv("Friendnet.csv",header=TRUE)
ProfMat<-read.csv("Profnet.csv",header=TRUE)
BossMat<-read.csv("BossNet.csv",header=TRUE)
SupportMat<-read.csv("SupportNet.csv",header=TRUE)

FriendMat<-as.matrix(FriendMat)
ProfMat<-as.matrix(ProfMat)
BossMat<-as.matrix(BossMat)
SupportMat<-as.matrix(SupportMat)


Friend.any <- ifelse(FriendMat > 0, 1, 0)
Boss.any <- ifelse(BossMat > 0, 1, 0)
Prof.any <- ifelse(ProfMat > 0, 1, 0)
Support.any <- ifelse(SupportMat > 0, 1, 0)

suppressPackageStartupMessages(library(igraph))

FriendGraph<-graph_from_adjacency_matrix(FriendMat, weighted=TRUE)
ProfGraph<-graph_from_adjacency_matrix(ProfMat, weighted=TRUE)
BossGraph<-graph_from_adjacency_matrix(BossMat, weighted=TRUE)
SupportGraph<-graph_from_adjacency_matrix(SupportMat, weighted=TRUE)

FriendGraph.any <-graph.adjacency(Friend.any,
                                  mode=c("directed"),
                                  weighted=NULL,
                                  diag=FALSE)
```

```
BossGraph.any <-graph.adjacency(Boss.any,
                                 mode=c("directed"),
                                 weighted=NULL,
                                 diag=FALSE)
ProfGraph.any <-graph.adjacency(Prof.any,
                                mode=c("directed"),
                                weighted=NULL,
                                diag=FALSE)
SupportGraph.any <-graph.adjacency(Support.any,
                                mode=c("directed"),
                                weighted=NULL,
                                diag=FALSE)
```

Remember, to create `network` objects, not graph objects, you'll need the `sna` package:

```
detach(package:igraph)
library(sna)
Friendnet<-as.network(Friend.any, directed=TRUE)
Bossnet<-as.network(Boss.any, directed=TRUE)
Profnet<-as.network(Prof.any, directed=TRUE)
Supportnet<-as.network(Support.any, directed=TRUE)
```

Now you have networks in both formats.

Working with data

Now we are ready to actually do something with our data. Of course, we usually have a research question, and let's pretend we have one. Questions 2:01-2:04, corresponding to Q30-Q33 in the Excel file, refer to the factor called "Physical participation" - how much does a person really put effort and energy into their job? It is a self-rated question, and of course, it's biased. But for now, we will not worry about biases as well as whether regression - the method we'll use today - is appropriate for analysis here. Actually, it's NOT appropriate, we have to build a factor-analytic model, but I'll let you explore that on your own, if you wish. For now, I am just showing the basics.

So let's pretend I am interested in learning how self-rated physical participation of each person depends on person's other characteristics (we've learned to call them attributes) as well as the networks they are embedded in.

The attributes file

Let's bring the file in. Please note that when we use the "header=TRUE" option, we can call variables by their names from the R environment. It's much more convenient than converting data to a matrix and referring to it in a matrix[,] format - we have close to 90 columns altogether. So keep an Excel file open, so you can check your variable names, when needed.

```
ocb_att<-read.csv('ocb_att.csv', header=TRUE)

# To make sure we got it right, let's look at the age variable:
ocb_att$Age
```

```
##  [1] 25 35 36 28 25 43 28 26 31 34 24 32 46 33 21 26 37 26 27 23 43 31 26 24 26
## [26] 24 38 51 29 23 23 25 24 31 28 26 35 24 43 27 22 28 25 24 29 20 26 23 30 27
## [51] 35 28 25 35 36 42 31 21 27 27 36 36 38 21 37 30 26 31
```

OK, looks good. Let's pull in the rest of the needed demographics. Also, what should we do with the "physical participation" factor? There are several ways to create a single variable out of it, most accurate

being the factor, but we do not have time to mess with factors today. So I will go ahead and just use the mean of the three items to create a single variable. Wrong, I know - but it's for demonstration purposes only (we can discuss why it's wrong and how to make it right some other time).

```
age<-ocb_att$Age
sex<-ocb_att$Sex
#How long the person had this position:
tenure<-ocb_att$WorkTitleYear+ocb_att$WorkTitleMonth/12
#How long worked in organization:
tenure_org<-ocb_att$WorkOrgYear+ocb_att$WorkOrgMonth/12
#How long reported to the same supervisor:
tenure_sup<-ocb_att$RepSupYear+ocb_att$RepSupMonth/12

# Set of dummies for education:
ed1<-ifelse(ocb_att$Education==3,1,0) # this is for secondary specialized
ed2<-ifelse(ocb_att$Education==4,1,0) # this is higher
# Secondary, obviously, is the baseline
```

Finally, the factor of physical participation, which we turn into a variable. To create an average, we use the "rowMeans" command, which calculates the mean of several columns in a row. When there is missing data (and there could be, since this is a live dataset), we need to account for it. For this purpose, we are using the "na.rm=TRUE" command, which tells us whether the missing values should be omitted from the calculations. Setting it to TRUE does omit them.

```
#Physical participation variable
phys_part<-ocb_att$Phys_Part
```

You can generate a large number of variables from your attributes file. But that's not even the most challenging part - the code below shows how easy it is to pull the rest of the variables.

```
HR_att<-read.csv("OCB_att.csv",header=TRUE)
#Let's get the sex of our respondents into its own vector:
sex<-HR_att$Sex
age<-HR_att$Age

#Dependent variables
Emotional_part<-HR_att$Emot_Part
Intent_to_leave<-HR_att$Intent_toLeave
Personal_conflicts<-HR_att$Personal_conflicts


#Predictors
#Challenge stressors:
Work_Quant<-HR_att$Work_quant #Work quantity
Work_Resp<-HR_att$Work_Resp #Work responsibility
Work_Diff<-HR_att$Work_Diff #work difficulty
Work_Speed<-HR_att$Work_Speed # Work speed
#Hindrance stressors:
Admin_problems<-HR_att$Admin_problems
Personal_conflicts<-HR_att$Personal_conflicts
```

Matching attributes to nodes and indices to actors

And now, the fun part starts - this is working with real data, right? Have you noticed that in our attributes file, we have 68 nodes, but in our networks - 122? Now, we just generated a bunch of network statistics, but that was for the entire network of 122 nodes, but how do we extract only those nodes for which we have

attributes? Until now, we've been given the nodes and the attributes in exactly the same order; today, we'll learn how to pass information with loops in R.

OK, so we have two sets of data:

1. Nodes that compose our network - the entire set of 122, on which we've calculated the network stats - these will become i nodes in our loops.
2. Nodes that have attributes (68 of them) - these will become j nodes in our loops.

We need predictor variables that are formed out of network statistics. We have 68 nodes that we are running the regression on, but we have 122 nodes for which we've obtained the network data. How do we know which nodes in 122 to take information from for our set of 68?

What we have to do is look through every node in set j, find a corresponding node in set i, and pass the attribute of the jth node to a variable in set i. In other words, we are using the network statistics, calculated on 122 nodes, but only creating vectors of variables for 68 nodes we have attributes for - otherwise, we can't use network attributes in regression. Now, if we want to pass the attributes from set j, such as gender, to our entire network of nodes i, we have to do it the other way around.

Believe it or not, we have to run the set of double loops to find matching nodes. This is because we loop through every node in set i and compare it with the FIRST node in set j; if we find a match between them, we grab it. Then, we have to loop again through every single node in set i for the SECOND node in set j, and so on until we've run through every node in the first set in an attempt to match it with a node from the second set.

Let's start with something simple, like gender. Let's say, we need to pass gender attribute to the network and create a picture of our network (to think of it, we haven't done it yet). The easiest way to match our nodes is on node names, because they are identical between all sets of data and networks.

First, create a vector that would hold the attribute (in this case, gender vector).

```
names<-ocb_att$Name # pull the names out of attributes dataset
gender_vector<-vector() #create a vector for gender
```

Next, run a double-loop and assign gender to a corresponding node:

```
suppressPackageStartupMessages(library(igraph))
for(i in 1:122){ # this is our set of all network nodes
  for(j in 1:68){ # this is our set of attribute-containing nodes
    # for each node in i, we run through all node in j
    # and compare names
    if(V(all_graph)$name[i]==names[j]){
      #if we match, we add the attribute to a vector
           gender_vector[i]<-sex[j]
           # and exit the inner loop for the next i
          break;}
    # if not, we are assigning a missing value
    # and keep going until we find a matching node
    else{gender_vector[i]<-NA}
  }
}

# Let's look at the result:
gender_vector
```

```
##   [1]  1  1  1  1  1  1  1  1  1  1  1  1  1  0  1  1  1  1  1  1  1  1  1  1
##  [26]  1  1  1  1  0  0  1  0  0  1  1  1  1  1  1  1  1  1  0  1  0  0  0  0
##  [51]  1  1  0  1  1  1  1  0  1  1  0  0  1  0  1 NA NA NA NA NA NA NA NA NA
```

```
##  [76] NA NA  1 NA NA NA NA NA NA NA NA NA  0 NA NA NA NA NA NA NA NA NA NA NA
## [101] NA NA NA NA  0 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

So here we are! Now all of our nodes have correct attributes. You can pass any other attributes you wish to nodes in exactly the same way.

Homework Assignment

And here is the actual assignment:

1. Choose any of the networks (friendship, professional, etc.) you wish.
2. Create a network and fit the unconditional ERGM model (edges only). Evaluate the result, calculate the probability of a tie formation.
3. Add the mutuality term, evaluate the result, calculate the probability of a tie formation.
4. Check for model degeneracy with mcmc.diagnostics()
5. Select some node-level attributes to include in your model; evaluate their significance and calculate the probabilities.
6. Include one or two main effects, evaluate them, calculate the probabilities.
7. Include at least one more complex structural term (e.g. gwesp). You may need to try a few different parameters until you find one that models triangles well in your data.
8. When you have a candidate model, simulate with gof(). Now you may find that your model is not a very good fit to the data, you will need to backtrack, maybe start by replacing your more complex structural term or add one more complex structural term.