Assignment 1

2024-10-28

——————————-TASK 1——————————- ——————————(1)——————————— After importing the data, and appropriately initializing the different adjacency matrices we build a simple QAP model for testing the relationship between the advice network (Id = 1) and the friendship network (Id = 2). Based on what we're asked to add on the subsequent parts of task 1, we're supposed to take the friendship network as the response network, and the advice network as the predictor network. After regression, we get the following results

	theta	exp(theta)	p-value
Intercept	-1.4987723	0.2234043	0.0160
advice	0.7255825	2.0659341	0.0284

The positive sign of the coefficient tied to the predictor network suggests that naming someone as friend increases the probability of asking them for advice. This suggestion can be seen to be of note by taking a look at the computed p-values. Both fall within the typical alpha level of 0.05. ——————————————(2)——————————— (i) Hypothesis we have to test: "Friendship nomination is more likely between pairs of managers within the same department." Clearly the adjacency matrix that we have to go with is adj_same_department which is a 21x21 matrix whose entry in row i and column j for i,j in {1,. . . ,21} is 1 whenever node i and node j are from the same department, and 0 otherwise.

(ii) Hypothesis we have to test: "Senior managers (<-> nodes with high"nodeTenure") are less likely to nominate friends" The idea is to have a matrix that for the entry corresponding to arc (i,j) stores the age of node i, since we need to test how senior managers nominate friends (not the other way around)

(iii) Hypothesis we have to test: "A friendship nomination is more likely between a pair of managers of a similar age." The idea here is simple enough, we just initialize a matrix that stores the information between differences in age. However the entries have to be "low" whenever the ages of the nodes is not similar, and high otherwise, so we use the following:

wght_similar_age[i,j] <- 1/(abs(all_nodes[i, "nodeAge"] - all_nodes[j, "nodeAge"])+1)

So we have wght_similar_age[i,j] = 1 whenever i and j have the same age, and a lower value as the age difference between i and j increases. ——————————(3)——————————— After performing MR-QAP for the model specified in (2) we get the following results:

	theta	exp(theta)	p-value
Intercept	-1.4987723	1.783013e-11	0.0000
advice	0.3655603	1.441321e+00	0.8942
same dep.	0.9038062	2.468983e+00	0.3714
seniority	42.1729747	2.067713e+18	0.9980
similar age	2.2283941	9.284943e+00	0.3280

All of those, except for the intercept hold very little statistical significance, as they do not fit within the alpha level of 0.05. Apparently there is too much noise for MR-QAP to find any meaningful relationship between

some of the predictor networks and the response network. ——————————(4)—————————— Another hypothesis we could test for is the following: "Everyone asks for advice to someone in the company that is in a higher position". We would need to make a weight matrix for the difference in level, and then use QAP as in the other cases. ——————————(5)—————————— The weight matrix described in (4) could be taken to be:

wght_level[i,j] <- all_nodes[j, "nodeLevel"] - all_nodes[i, "nodeLevel"]

so wght_leve[i,j] is positive whenever j is at a higher level, and it is negative if the opposite holds. However, as there's no reason in having a higher weight if the difference is higher, it's better to take the maximum of that difference with the number one. So we take:

wght_level[i,j] <- max(all_nodes[j, "nodeLevel"] - all_nodes[i, "nodeLevel"],1)

	theta	exp(theta)	p-value
Intercept	-22.9972117	1.029053e-10	0.0212
advice	0.3473968	1.415378e+00	0.8924
same dep.	0.8489238	2.337130e+00	0.3906
seniority	42.1422672	2.005183e+18	0.9978
similar age	1.0851201	2.959795e+00	0.3506
lvl. diff.	-1.6584837	1.904275e-01	0.7994

Again, the high p values don't really allow us to see anything but noise. Even though the p-value for the intercept is relevant, the fact that it significantly changed from the previous model could be an indication that the level difference matrix added more noise than anything.

——————————-TASK 2——————————-

```
# Network Modeling - HS 2024
# C. Stadtfeld, A. Uzaheta and I. Smokovic
# Social Networks Lab
# Department of Humanities, Social and Political Sciences
# ETH Zurich
# 14 October 2024
#
# Assignment 1 - Task 2


# MHstep ------------------------------------------------------------------
```

Simulate the next step of a network in Markov chain using Metropolis-Hasting

The function `MHstep` simulates the the Metropolis-Hastings step that defines the Markov chain whose stationary distribution is the ERGM with edge, mutual and nodematch statistics

@param net an object of class `matrix`. Adjacency matrix of the network. @param theta1 a numeric value. The value of the edge parameter of the ERGM. @param theta2 a numeric value. The value of the mutual parameter of the ERGM. @param theta3 a numeric value. The value of the istar(2) parameter of the ERGM.

@return next state of the Markov Chain

@examples MHstep(matrix(c(0, 1, 0, 0, 0, 0, 1, 1, 0), nrow = 3, ncol = 3), -log(0.5), log(0.4), log(.8))

```
MHstep <- function(net, theta1, theta2, theta3){

  # Number of vertices in the network
  nvertices <- nrow(net)
```

```
  # Choose randomly two vertices, prevent loops {i,i} with replace = FALSE
  tie <- sample(1:nvertices, 2, replace = FALSE)
  i <- tie[1]
  j <- tie[2]

  # Compute the change statistics
  #                 --- MISSING---
  # we toggle i->j:
  delta_edges <- ifelse(net[i,j] == 1, -1, 1)  # if there is an edge from i->j, we remove it otherwise
  delta_mutual <- ifelse(net[i,j] == 0, net[j,i], -net[j,i]) # mathematically, we have delta_mutual = x
  delta_in2stars <- 0  # the indegree x_{+i} for node i is not affected, since we only toggle i->j

  # Compute the probability of the next state
  # according to the Metropolis-Hastings algorithm
  #                 --- MISSING---
  p = min(1, exp(theta1*delta_edges + theta2*delta_mutual + theta3*delta_in2stars)) # this is the formu

  # Select the next state:
  #                 --- MISSING---
  # This choice is somewhat arbitrary: We toggle i->j, if the computed probability p is larger than a g
  if (runif(1) <= p) {
    # Toggle the tie
    net[i, j] <- 1 - net[i,j]
  }

  # Return the next state of the chain
  return(net)
}
# sanity check
# MHstep(matrix(c(0, 1, 0, 0, 0, 0, 1, 1, 0), nrow = 3, ncol = 3), -log(0.5), log(0.4), log(.8))

# Markov Chain simulation ------------------------------------------------
```

The function MarkovChain simulates the networks from the ERGM with edge, mutual and nodematch statistics

@param net an object of class `matrix`. Adjacency matrix of the network. @param theta1 a numeric value. The value of the edge parameter of the ERGM. @param theta2 a numeric value. The value of the mutual parameter of the ERGM. @param theta3 a numeric value. The value of the istar(2) parameter of the ERGM. @param burnin an integer value. Number of steps to reach the stationary distribution. @param thinning an integer value. Number of steps between simulated networks. @param nNet an integer value. Number of simulated networks to return as output.

@return a named list: - netSim: an `array` with the adjancency matrices of the simulated networks. - statSim: a `matrix` with the value of the statistic defining the ERGM.

@examples MarkovChain(matrix(c(0, 1, 0, 0, 0, 0, 1, 1, 0), nrow = 3, ncol = 3), -log(0.5), log(0.4), log(.8))

```
MarkovChain <- function(
    net,
    theta1, theta2, theta3,
    burnin = 10000, thinning = 1000, nNet = 1000){

  # Burnin phase: repeating the steps of the chain "burnin" times
  nvertices <- nrow(net)
  burninStep <- 1 # counter for the number of burnin steps
```

```r
  # Perform the burnin steps
  #                 --- MISSING---
  # We let the random walk run for a long time to make sure that it does not depned on the starting net
  for (i in 1:burnin) {
    net <- MHstep(net, theta1, theta2, theta3)
  }

  # After the burnin phase we draw the networks
  # The simulated networks and statistics are stored in the objects
  # netSim and statSim
  netSim <- array(0, dim = c(nvertices, nvertices, nNet))
  statSim <- matrix(0, nNet, 3)
  thinningSteps <- 0 # counter for the number of thinning steps
  netCounter <- 1 # counter for the number of simulated network

  #                 --- MISSING---
  while (netCounter <= nNet){
    # compute the new network:
    net <- MHstep(net, theta1, theta2, theta3)

    # compute the new statistics:
    edges <- sum(net)
    dyads <- sum(net * t(net) * upper.tri(net))
    indegree <- colSums(net)
    in2stars <- sum(choose(indegree, 2))

    # update:
    netSim[,,netCounter] <- net
    statSim[netCounter,] <- c(edges, dyads, in2stars)
    netCounter <- netCounter + 1

    # thinning: simulate thinning many networks and throw them away
    for (i in 1:thinning){
      net <- MHstep(net, theta1, theta2, theta3)
    }
  }

  # Return the simulated networks and the statistics
  return(list(netSim = netSim, statSim = statSim))
}
#####################################################
#################### part (2) ####################

observed_net <- matrix(c(0, 1, 0, 0, 0, 0, 1, 1, 0), nrow = 3, ncol = 3)

# statistics of the observed network:
edges <- sum(observed_net)
dyads <- sum(observed_net * t(observed_net) * upper.tri(observed_net))
indegree <- colSums(observed_net)
in2stars <- sum(choose(indegree, 2))
observed_stats <- c(edges, dyads, in2stars)

# suggested parameters:
```

```r
theta <- c(-2.76, 0.68, 0.05 )

# simulate the MarkovChain:
list_simulation <- MarkovChain(observed_net, theta[1], theta[2], theta[3])

# analyse the moment equation to see if the parameters are reasonable
sample_equivalent <- colMeans(list_simulation$statSim)  # this is the formulate on slide 22
moment_eq <- sample_equivalent - observed_stats         # this should be closed to 0 if the parameters
print(paste("Values of the moment equation for the given parameters:", paste(moment_eq, collapse = ", "
```

```
## [1] "Values of the moment equation for the given parameters: -2.636, 0.015, -0.99"
```

Conclusion: The values of the moments equation should be close to 0 if the parameters are a good choice. This is not the case. Hence, we should search for better parameters.

```r
#####################################################
#################### part (3) #####################
```

We update the parameters in virtue of the Robbins-Monro algorithm on slide 23:

```r
num_it = 5  # here I tried different values in {1,...,10}
for (n in 1:num_it){
  a = 1/n^2   # this needs to be a sequence converging to 0
  theta <- theta - a*moment_eq # update via formula on slide 23 (with D = I)
  # simulate the MarkovChain and compute the value of the moment equation
  list_simulation <- MarkovChain(observed_net, theta[1], theta[2], theta[3])
  sample_equivalent <- colMeans(list_simulation$statSim)
  moment_eq <- sample_equivalent - observed_stats
}

# check if the new parameters are better:
list_simulation <- MarkovChain(observed_net, theta[1], theta[2], theta[3])
sample_equivalent <- colMeans(list_simulation$statSim)  # this is the formulate on slide 22
moment_eq <- sample_equivalent - observed_stats         # this should be closed to 0 if the parameters
print(paste("Values of the moment equation for the updated parameters:", paste(moment_eq, collapse = ",
```

```
## [1] "Values of the moment equation for the updated parameters: -0.14, 0.724, -0.313"
```

```r
print(paste("Updated theta:", paste(theta, collapse = ", ")))
```

```
## [1] "Updated theta: -0.188761944444444, 0.240991111111111, 1.12848055555556"
```

Conclusion: As we can see, the updated parameters yield a smaller value of the moment equation. Since we want the value of the moment equation to be close to 0, the new parameters are better than the previous ones.

Task 3

```r
# install.packages(c("robustbase"), lib="~/R/library", repos = "https://cloud.r-project.org/")
# install.packages(c("sna","network","ergm","igraph"), lib="~/R/library", repos = "https://cloud.r-proj

library(sna)
library(network)
library(ergm)

nodes <- read.table("Dataset/Krackhardt-High-Tech_nodes.txt", header = TRUE)
```

```r
edges <- read.table("Dataset/Krackhardt-High-Tech_multiplex.edges", header = TRUE)
layers <- read.table("Dataset/Krackhardt-High-Tech_layers.txt", header = TRUE)

head(nodes)
```

```
##   nodeID nodeAge nodeTenure nodeLevel nodeDepartment
## 1      1      33          9         3              4
## 2      2      42         20         2              4
## 3      3      40         13         3              2
## 4      4      33          8         3              4
## 5      5      32          3         3              2
## 6      6      59         28         3              1
```

```r
head(edges)
```

```
##   layerID nodeID1 nodeID2 weight
## 1       1       1       2      1
## 2       1       1       4      1
## 3       1       1       8      1
## 4       1       1      16      1
## 5       1       1      18      1
## 6       1       1      21      1
```

```r
layer_to_analyze <- 2
layer_edges <- edges[edges$layerID == layer_to_analyze,]

net <- network(layer_edges[, c("nodeID1", "nodeID2")], directed = TRUE)

net%v%"age" <- nodes$nodeAge
net%v%"tenure" <- nodes$nodeTenure
net%v%"level" <- nodes$nodeLevel
net%v%"department" <- nodes$nodeDepartment
cat("Network for Layer:", layer_to_analyze, "\n")
```

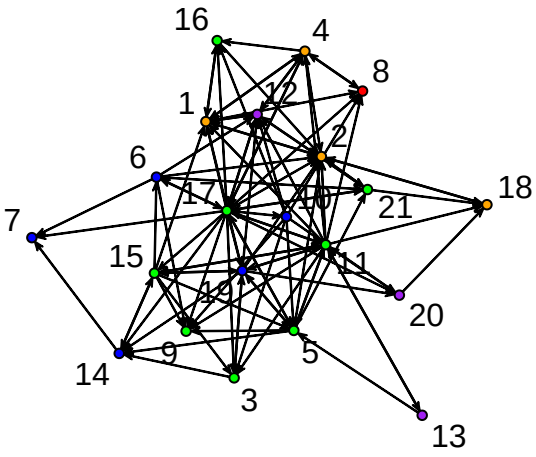
```
## Network for Layer: 2
```

```r
print(net)
```

```
##  Network attributes:
##   vertices = 21
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 102
##     missing edges= 0
##     non-missing edges= 102
##
##  Vertex attribute names:
##     age department level tenure vertex.names
##
## No edge attributes
```

```r
dept_colors <- c("0" = "red",      # CEO
                 "1" = "blue",
```

```
                  "2" = "green",
                  "3" = "purple",
                  "4" = "orange")

vertex_colors <- dept_colors[as.character(get.vertex.attribute(net, "department"))]

plot(net, vertex.col = vertex_colors, displaylabels = TRUE,
     main = paste("Network Visualization -", layers$layerLabel[layers$layerID == layer_to_analyze]))
```




```
set.seed(1984) #literally 1984
```

Task 3.1

```
model0 <- ergm(net ~ edges + nodematch("department"))
```

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

```
summary(model0)
```

```
## Call:
## ergm(formula = net ~ edges + nodematch("department"))
##
## Maximum Likelihood Results:
##
##                      Estimate Std. Error MCMC % z value Pr(>|z|)
## edges                 -1.2786     0.1342      0  -9.528   <1e-04 ***
## nodematch.department   0.5694     0.2572      0   2.214   0.0268 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 582.2  on 420  degrees of freedom
##  Residual Deviance: 460.9  on 418  degrees of freedom
##
## AIC: 464.9  BIC: 473  (Smaller is better. MC Std. Err. = 0)
```

```
theta <- coef(model0)

p <- exp(sum(theta)) / (1 + exp(sum(theta)))
```

The conditional probability of observing a tie between i and j assuming they work in the same department is:

```
p
```

```
## [1] 0.3297872
```

Task 3.2

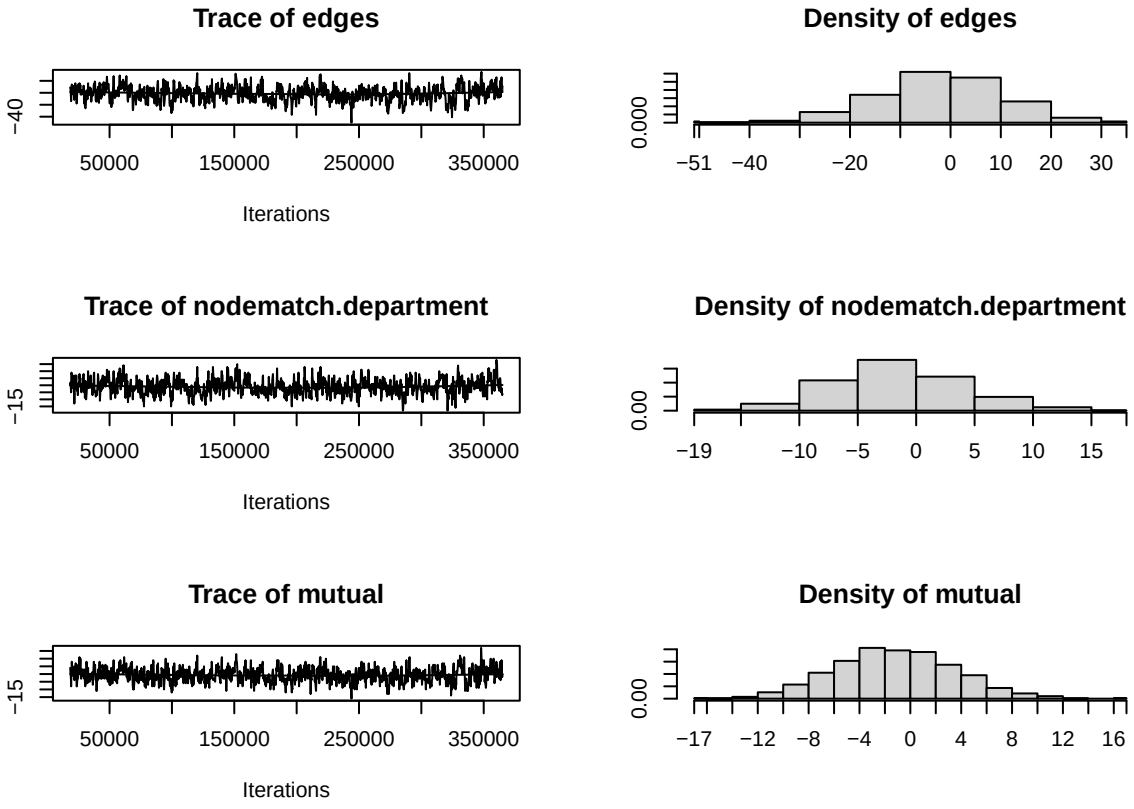
```
model1 <- ergm(net ~ edges
               + nodematch("department")
               + mutual                                  # (i)
               + gwesp(decay = 0.3, fixed = TRUE)        # (ii)
               + gwidegree(decay = 0.3, fixed = TRUE))   # (iii)
```

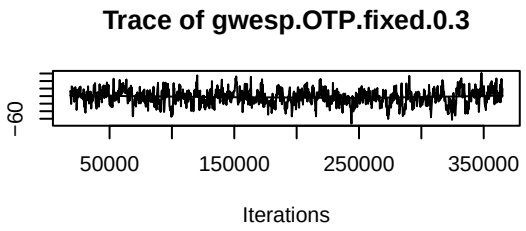
```
## Starting maximum pseudolikelihood estimation (MPLE):

## Obtaining the responsible dyads.

## Evaluating the predictor and response matrix.

## Maximizing the pseudolikelihood.

## Finished MPLE.

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

## Iteration 1 of at most 60:

## 1 Optimizing with step length 0.9058.
## The log-likelihood improved by 3.1201.
## Estimating equations are not within tolerance region.
## Iteration 2 of at most 60:
## 1 Optimizing with step length 1.0000.
## The log-likelihood improved by 0.1617.
## Estimating equations are not within tolerance region.
## Iteration 3 of at most 60:
## 1 Optimizing with step length 1.0000.
## The log-likelihood improved by 0.0375.
## Convergence test p-value: 0.6470. Not converged with 99% confidence; increasing sample size.
```

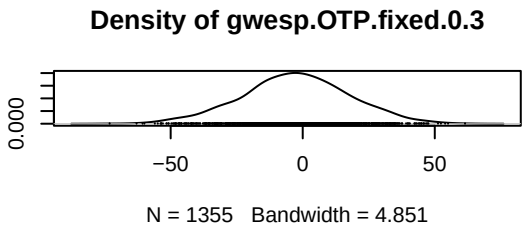
```
## Iteration 4 of at most 60:
## 1 Optimizing with step length 1.0000.
## The log-likelihood improved by 0.0508.
## Convergence test p-value: 0.5707. Not converged with 99% confidence; increasing sample size.
## Iteration 5 of at most 60:
## 1 Optimizing with step length 1.0000.
## The log-likelihood improved by 0.1291.
## Estimating equations are not within tolerance region.
## Estimating equations did not move closer to tolerance region more than 1 time(s) in 4 steps; increas
## Iteration 6 of at most 60:
## 1 Optimizing with step length 1.0000.
## The log-likelihood improved by 0.0239.
## Convergence test p-value: 0.0003. Converged with 99% confidence.
## Finished MCMLE.
## Evaluating log-likelihood at the estimate. Fitting the dyad-independent submodel...
## Bridging between the dyad-independent submodel and the full model...
## Setting up bridge sampling...
## Using 16 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 .
## Bridging finished.
##
## This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the
## mcmc.diagnostics() function.
```

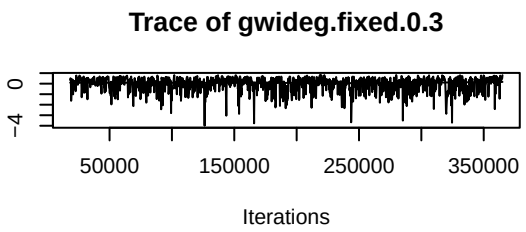
Task 3.3

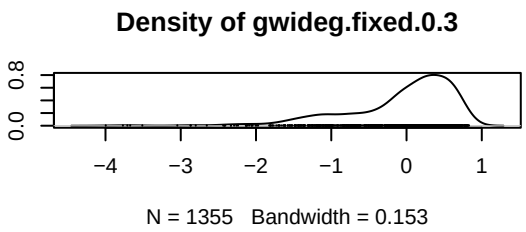
```
mcmc.diagnostics(model1)
```












```
## Sample statistics summary:
##
## Iterations = 18432:365056
## Thinning interval = 256
## Number of chains = 1
## Sample size per chain = 1355
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                           Mean      SD Naive SE Time-series SE
## edges                -1.52620 12.3809   0.3363        0.75035
## nodematch.department -1.07675  5.5094   0.1497        0.29798
## mutual               -0.73579  4.6959   0.1276        0.24695
## gwesp.OTP.fixed.0.3  -2.64506 20.1939   0.5486        1.20453
## gwideg.fixed.0.3     -0.06928  0.7177   0.0195        0.02898
##
## 2. Quantiles for each variable:
##
##                          2.5%      25%     50%     75%   97.5%
## edges                -27.000  -9.0000 -1.0000  6.0000 22.1500
## nodematch.department -11.000  -5.0000 -1.0000  3.0000 10.1500
## mutual               -10.000  -4.0000 -1.0000  2.0000  9.0000
## gwesp.OTP.fixed.0.3  -44.285 -15.3793 -2.7356 10.5658 35.4813
## gwideg.fixed.0.3      -1.789  -0.3734  0.1473  0.4446  0.7195
##
```

```
##
## Are sample statistics significantly different from observed?
##                   edges nodematch.department       mutual gwesp.OTP.fixed.0.3 gwideg.fixed.0.3
## diff.      -1.52619926        -1.0767527675 -0.735793358         -2.64506426      -0.06927528
## test stat. -2.03398007        -3.6134821871 -2.979477291         -2.19593867      -2.39036630 20.4205
## P-val.      0.04195361         0.0003021121  0.002887406          0.02809633       0.01683158  0.0011
##
## Sample statistics cross-correlations:
##                           edges nodematch.department    mutual gwesp.OTP.fixed.0.3 gwideg.fixed.0.3
## edges                 1.0000000            0.5601179 0.7990950           0.9718258        0.5573484
## nodematch.department 0.5601179            1.0000000 0.4892555           0.5513304        0.3243013
## mutual               0.7990950            0.4892555 1.0000000           0.8190854        0.4220245
## gwesp.OTP.fixed.0.3  0.9718258            0.5513304 0.8190854           1.0000000        0.4800251
## gwideg.fixed.0.3     0.5573484            0.3243013 0.4220245           0.4800251        1.0000000
##
## Sample statistics auto-correlation:
## Chain 1
##              edges nodematch.department    mutual gwesp.OTP.fixed.0.3 gwideg.fixed.0.3
## Lag 0    1.0000000            1.0000000 1.0000000           1.0000000       1.00000000
## Lag 256  0.6651735            0.5968492 0.5395621           0.6186680       0.25646431
## Lag 512  0.4540913            0.3606875 0.3309684           0.4215240       0.14213639
## Lag 768  0.3300527            0.2396933 0.2242273           0.3050133       0.10169983
## Lag 1024 0.2282775            0.1715241 0.1615200           0.2110624       0.07111984
## Lag 1280 0.1607653            0.1023366 0.1335552           0.1433232       0.03733592
##
## Sample statistics burn-in diagnostic (Geweke):
## Chain 1
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##                edges nodematch.department               mutual  gwesp.OTP.fixed.0.3     gwideg.fixed
##             1.413142             1.858897             1.301284             1.323001             1.634
##
## Individual P-values (lower = worse):
##                edges nodematch.department               mutual  gwesp.OTP.fixed.0.3     gwideg.fixed
##           0.15761394           0.06304171           0.19316112           0.18583502           0.1021
## Joint P-value (lower = worse):  0.8156918
##
## Note: MCMC diagnostics shown here are from the last round of simulation, prior to computation of fina
##   parameter estimates. Because the final estimates are refinements of those used for this simulation
##   these diagnostics may understate model performance. To directly assess the performance of the final
##   on in-model statistics, please use the GOF command: gof(ergmFitObject, GOF=~model).
```

In the MCDC diagnostic we can look at the traceplots for the different terms. They describe the deviation of the statistic value in each sampled network from the observed value, on the right we have the distribution of the sample statistic deviations. As they describe in *here* one should look for a "fuzzy caterpillar" shape in the traceplot which we have for all terms except for the last one which indicates good mixing and stability. In the last term there are some strong outliers into the negative but there is no trend away from the observed value. Also the autocorrelation falls off quickly for each term so we conclude that the model is stable and converged nicely.

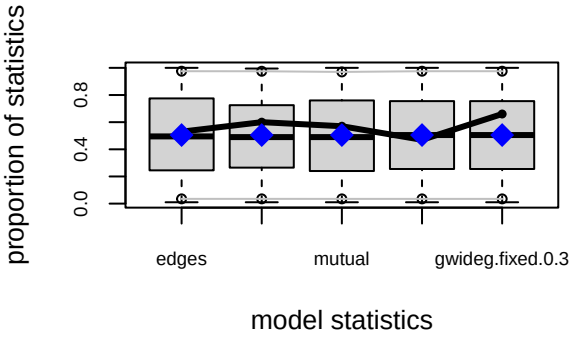
Task 3.4

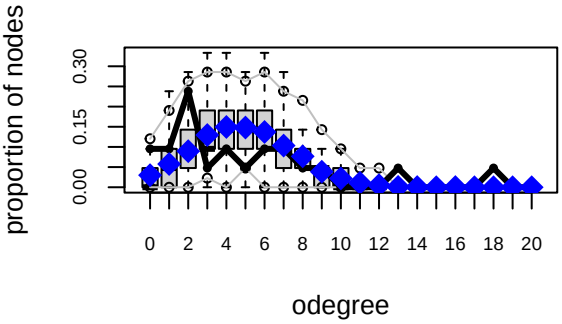
```
model1gof <- gof(model1)
model1gof
```

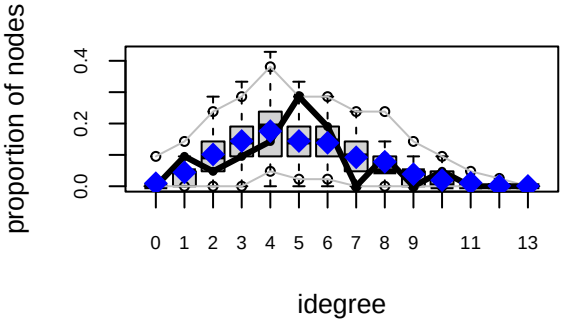
```
##
## Goodness-of-fit for in-degree
##
##           obs min mean max MC p-value
## idegree0    0   0 0.20   2       1.00
## idegree1    2   0 0.93   3       0.48
## idegree2    1   0 2.13   6       0.78
## idegree3    2   0 3.03   9       0.80
## idegree4    3   0 3.69   9       0.98
## idegree5    6   0 3.03   8       0.14
## idegree6    4   0 2.91   8       0.64
## idegree7    0   0 1.94   5       0.36
## idegree8    2   0 1.64   8       0.92
## idegree9    0   0 0.81   3       0.88
## idegree10   1   0 0.41   2       0.72
## idegree11   0   0 0.23   3       1.00
## idegree12   0   0 0.03   1       1.00
## idegree13   0   0 0.02   1       1.00
##
## Goodness-of-fit for out-degree
##
##           obs min mean max MC p-value
## odegree0    2   0 0.63   3       0.26
## odegree1    2   0 1.21   5       0.70
## odegree2    5   0 1.88   7       0.08
## odegree3    1   0 2.72   7       0.42
## odegree4    2   0 3.14   7       0.72
## odegree5    1   0 3.11   6       0.30
## odegree6    2   0 2.88   7       0.80
## odegree7    2   0 2.17   6       1.00
## odegree8    1   0 1.61   6       1.00
## odegree9    1   0 0.81   4       1.00
## odegree10   0   0 0.46   3       1.00
## odegree11   0   0 0.23   2       1.00
## odegree12   0   0 0.11   1       1.00
## odegree13   1   0 0.02   1       0.04
## odegree14   0   0 0.01   1       1.00
## odegree16   0   0 0.01   1       1.00
## odegree18   1   0 0.00   0       0.00
##
## Goodness-of-fit for edgewise shared partner
##
##          obs min  mean max MC p-value
## esp.OTP0  18   5 13.81  28       0.42
## esp.OTP1  20  21 36.51  51       0.00
## esp.OTP2  29  11 30.69  53       0.94
## esp.OTP3  14   2 14.68  33       1.00
## esp.OTP4  15   0  5.34  20       0.10
## esp.OTP5   1   0  1.41   9       1.00
```

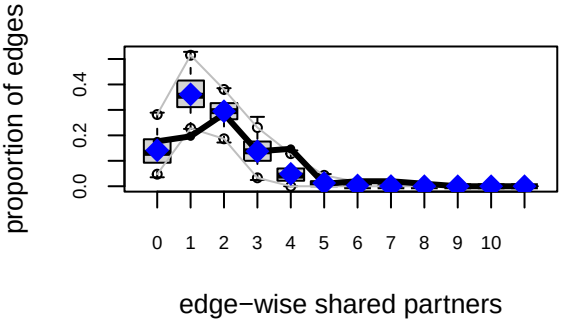
```
## esp.OTP6   2   0  0.36   4       0.12
## esp.OTP7   2   0  0.08   3       0.02
## esp.OTP8   1   0  0.01   1       0.02
##
## Goodness-of-fit for minimum geodesic distance
##
##     obs min   mean max MC p-value
## 1   102  64 102.89 142       1.00
## 2   168 128 201.58 254       0.22
## 3    81  33  83.47 125       0.88
## 4    27   0  12.59  50       0.22
## 5     2   0   1.32  16       0.50
## 6     0   0   0.22  14       1.00
## 7     0   0   0.10  10       1.00
## 8     0   0   0.02   2       1.00
## Inf  40   0  17.81  77       0.44
##
## Goodness-of-fit for model statistics
##
##                            obs      min      mean       max MC p-value
## edges                  102.0000 64.00000 102.89000 142.00000       1.00
## nodematch.department    31.0000 17.00000  32.27000  46.00000       0.92
## mutual                  23.0000 10.00000  23.20000  37.00000       1.00
## gwesp.OTP.fixed.0.3    103.3383 40.17384 104.30928 168.14427       0.94
## gwideg.fixed.0.3        27.4802 24.85895  27.45863  28.32429       0.68
```

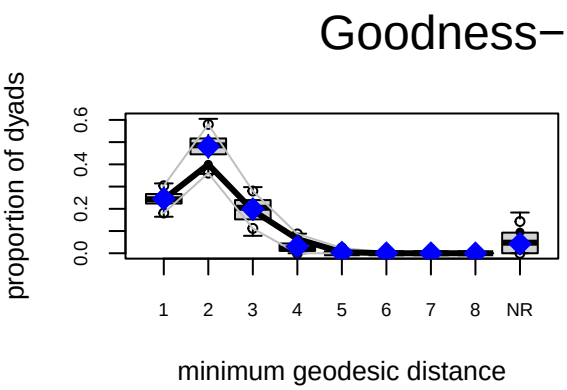
```
par(mfrow=c(2,2),mar=c(5, 4, 4, 2))
plot(model1gof)
```


proportion of statistics
0.0
0.4
0.8
edges
mutual
gwideg.fixed.0.3
model statistics


proportion of nodes
0.00
0.15
0.30
0 2 4 6 8 10 12 14 16 18 20
odegree


proportion of nodes
0.0
0.2
0.4
0 1 2 3 4 5 6 7 8 9 11 13
idegree


proportion of edges
0.0
0.2
0.4
0 1 2 3 4 5 6 7 8 9 10
edge−wise shared partners



The plot shows that there is no significant difference between the average simulation and the observed value as the black lines (observed values) almost never cross the gray confidence interval. There are two exceptions for odegree at value 18 and wide-wise shared partners at value 4. Overall criteria, the observed network falls close to the center of the gray regions of the simulation distribution thus the model is a good fit.

```
summary(model1)
```

```
## Call:
## ergm(formula = net ~ edges + nodematch("department") + mutual +
##     gwesp(decay = 0.3, fixed = TRUE) + gwidegree(decay = 0.3,
##     fixed = TRUE))
##
## Monte Carlo Maximum Likelihood Results:
##
##                       Estimate Std. Error MCMC % z value Pr(>|z|)
## edges                  -3.1169     0.3837      0  -8.122  < 1e-04 ***
## nodematch.department    0.4146     0.2203      0   1.882  0.05987 .
## mutual                  0.9911     0.3732      0   2.656  0.00792 **
## gwesp.OTP.fixed.0.3     0.9170     0.2324      0   3.947  < 1e-04 ***
## gwideg.fixed.0.3        1.5669     1.7725      0   0.884  0.37669
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 582.2  on 420  degrees of freedom
##  Residual Deviance: 429.1  on 415  degrees of freedom
##
## AIC: 439.1  BIC: 459.3  (Smaller is better. MC Std. Err. = 0.4501)
```

$\alpha = 0.01$

Edges: The parameter is significant and negative thus the network is sparse.

Department homophily: The parameter is not significant thus the data doesn't support that ties are more likely within a department.

Mutal: The parameter is significant and positive thus it indicates evidence for reciprocity.

Transitivity: The parameter is significant and positive thus it indicates evidence for transitivity.

Popularity: The parameter is not significant thus the data doesn't show that ties are more likely when the receiver has a higher in-degree.

Task 4

Task 4.1

```
model2 <- ergm(net ~ edges
               + nodematch("department")
               + nodeocov("tenure")
               + absdiff("age"))
```

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

## Obtaining the responsible dyads.

## Evaluating the predictor and response matrix.

## Maximizing the pseudolikelihood.

## Finished MPLE.

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

```
summary(model2)
```

```
## Call:
## ergm(formula = net ~ edges + nodematch("department") + nodeocov("tenure") +
##     absdiff("age"))
##
## Maximum Likelihood Results:
##
##                      Estimate Std. Error MCMC % z value Pr(>|z|)
## edges                -0.93463    0.26161      0  -3.573 0.000353 ***
## nodematch.department  0.45022    0.26311      0   1.711 0.087056 .
## nodeocov.tenure      -0.05512    0.01706      0  -3.230 0.001236 **
## absdiff.age           0.02626    0.01322      0   1.987 0.046916 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 582.2  on 420  degrees of freedom
##  Residual Deviance: 446.6  on 416  degrees of freedom
##
## AIC: 454.6  BIC: 470.7  (Smaller is better. MC Std. Err. = 0)
```

```
model3 <- ergm(net ~ edges
               + nodematch("department")
               + nodeocov("tenure")
               + absdiff("age")
```

```
              + mutual
              + gwesp(decay = 0.3, fixed = TRUE)
              + gwidegree(decay = 0.3, fixed = TRUE))
```

```
## Starting maximum pseudolikelihood estimation (MPLE):
## Obtaining the responsible dyads.
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Finished MPLE.
## Starting Monte Carlo maximum likelihood estimation (MCMLE):
## Iteration 1 of at most 60:
## 1 Optimizing with step length 1.0000.
## The log-likelihood improved by 1.9001.
## Estimating equations are not within tolerance region.
## Iteration 2 of at most 60:
## 1 Optimizing with step length 1.0000.
## The log-likelihood improved by 0.0994.
## Convergence test p-value: 0.4305. Not converged with 99% confidence; increasing sample size.
## Iteration 3 of at most 60:
## 1 Optimizing with step length 1.0000.
## The log-likelihood improved by 0.0418.
## Convergence test p-value: 0.6322. Not converged with 99% confidence; increasing sample size.
## Iteration 4 of at most 60:
## 1 Optimizing with step length 1.0000.
## The log-likelihood improved by 0.0778.
## Convergence test p-value: 0.0274. Not converged with 99% confidence; increasing sample size.
## Iteration 5 of at most 60:
## 1 Optimizing with step length 1.0000.
## The log-likelihood improved by 0.0120.
## Convergence test p-value: 0.0480. Not converged with 99% confidence; increasing sample size.
## Iteration 6 of at most 60:
## 1 Optimizing with step length 1.0000.
## The log-likelihood improved by 0.0042.
## Convergence test p-value: 0.0043. Converged with 99% confidence.
## Finished MCMLE.
## Evaluating log-likelihood at the estimate. Fitting the dyad-independent submodel...
## Bridging between the dyad-independent submodel and the full model...
## Setting up bridge sampling...
## Using 16 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 .
## Bridging finished.
##
## This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the
## mcmc.diagnostics() function.
```

```
summary(model3)
```

```
## Call:
## ergm(formula = net ~ edges + nodematch("department") + nodeocov("tenure") +
##     absdiff("age") + mutual + gwesp(decay = 0.3, fixed = TRUE) +
##     gwidegree(decay = 0.3, fixed = TRUE))
##
## Monte Carlo Maximum Likelihood Results:
##
##                      Estimate Std. Error MCMC % z value Pr(>|z|)
```

```
## edges                -2.62408    0.46765      0  -5.611  < 1e-04 ***
## nodematch.department  0.27236    0.23572      0   1.155  0.24791
## nodeocov.tenure      -0.03576    0.01472      0  -2.430  0.01511 *
## absdiff.age           0.01644    0.01035      0   1.589  0.11217
## mutual                1.01390    0.38695      0   2.620  0.00879 **
## gwesp.OTP.fixed.0.3   0.77775    0.24777      0   3.139  0.00170 **
## gwideg.fixed.0.3      1.10591    1.95020      0   0.567  0.57066
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 582.2  on 420  degrees of freedom
##  Residual Deviance: 419.3  on 413  degrees of freedom
##
## AIC: 433.3  BIC: 461.6  (Smaller is better. MC Std. Err. = 0.4789)
```

Task 4.2

Hypothesis that can be tested with ERGMs but not with MR-QAPs: H_0: "There are triangles in the advice network." In order to test this hypothesis with an ERGM we need to include the following statistics:

$$z_m(x) = \sum_{j,k,l} x^a_{jk} x^a_{jl} x^a_{lk},$$

where $(x_{ij})_{ij}$ denotes the adjacency matrix of the advice network.