

# Introduction to Exponential-family Random Graph (ERG or $p^*$ ) modeling with *ergm*

Version 4.0-3887

The Statnet Development Team

March 23, 2018

## Contents

<b>1</b>	<b>Getting Started</b>	<b>1</b>
<b>2</b>	<b>Statistical network modeling; the <i>ergm</i> command and <i>ergm</i> object</b>	<b>2</b>
<b>3</b>	<b>Model terms available for <i>ergm</i> estimation and simulation</b>	<b>11</b>
3.1	Terms provided with <i>ergm</i> . . . . .	11
3.2	Coding new terms . . . . .	11
<b>4</b>	<b>Network simulation: the <i>simulate</i> command and <i>network.list</i> objects</b>	<b>11</b>
<b>5</b>	<b>Examining the quality of model fit – <i>GOF</i></b>	<b>13</b>
<b>6</b>	<b>Diagnostics: troubleshooting and checking for model degeneracy</b>	<b>18</b>
<b>7</b>	<b>Working with egocentrically sampled network data</b>	<b>33</b>
<b>8</b>	<b>Additional functionality in the statnet family of packages</b>	<b>46</b>
8.1	In the <i>ergm</i> and <i>network</i> packages: . . . . .	46
8.2	In other packages from the statnet suite: . . . . .	46
8.2.1	Static (cross-sectional) network analysis packages . . . . .	46
8.2.2	Dynamic (longitudinal) network analysis packages: . . . . .	46
8.3	R packages that build on statnet . . . . .	46
8.4	Additional functionality in development: . . . . .	47
<b>9</b>	<b>Statnet Commons: The core development team</b>	<b>47</b>

## 1 Getting Started

This vignette is based on the *ergm* tutorial presented at INSNA Sunbelt - St. Pete Beach, Florida, Feb 2011.

Open an R session, and set your working directory to the location where you would like to save this work. You can do this with the pull-down menus (File¿Change Dir) or with the command:

```
setwd('full.path.for.the.folder')
```

To install all of the packages in the statnet suite:

```
install.packages('statnet')
library(statnet)
```

Or, to only install the specific statnet packages needed for this tutorial:

```
install.packages('network')
install.packages('ergm')
install.packages('sna')
library(network)
library(ergm)
library(sna)
```

After the first time, to update the packages one can either repeat the commands above, or use:

```
update.packages('name.of.package')
```

For this tutorial, we will need one additional package (coda), which is recommended (but not required) by ergm:

```
install.packages('coda')
library(coda)
```

## 2 Statistical network modeling; the *ergm* command and *ergm* object

Make sure the statnet package is attached:

```
library(statnet)
```

or

```
library(ergm)

## Loading required package: network
## network: Classes for Relational Data
## Version 1.14-335 created on 2018-03-14.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##               Mark S. Handcock, University of California -- Los Angeles
##               David R. Hunter, Penn State University
##               Martina Morris, University of Washington
##               Skye Bender-deMoll, University of Washington
## For citation information, type citation("network").
## Type help("network-package") to get started.
```

```

##
## ergm: version 4.0-3887, created on 2018-03-23
## Copyright (c) 2018, Mark S. Handcock, University of California -- Los Angeles
##          David R. Hunter, Penn State University
##          Carter T. Butts, University of California -- Irvine
##          Steven M. Goodreau, University of Washington
##          Pavel N. Krivitsky, University of Wollongong
##          Martina Morris, University of Washington
##          with contributions from
##          Li Wang
##          Kirk Li, University of Washington
##          Skye Bender-deMoll, University of Washington
## Based on "statnet" project software (statnet.org).
## For license and citation information see statnet.org/attribution
## or type citation("ergm").

## NOTE: Versions before 3.6.1 had a bug in the implementation of the
## bd() constraint which distorted the sampled distribution somewhat.
## In addition, Sampson's Monks datasets had mislabeled vertices. See
## the NEWS and the documentation for more details.

library(sna)

## Loading required package: statnet.common
##
## Attaching package: 'statnet.common'
## The following object is masked from 'package:base':
##
##   order
##
## sna: Tools for Social Network Analysis
## Version 2.4 created on 2016-07-23.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
## For citation information, type citation("sna").
## Type help(package="sna") to get started.

set.seed(1)

```

The ergm package contains several network data sets that you can use for practice examples.

```

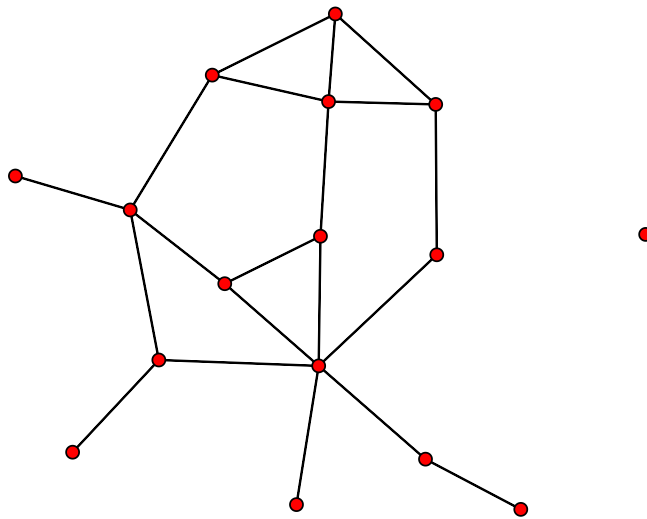
data(package='ergm') # tells us the datasets in our packages
data(florentine) # loads flomarriage and flobusiness data
flomarriage # Let's look at the flomarriage data

## Network attributes:
##   vertices = 16
##   directed = FALSE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 20
##   missing edges= 0

```

```
##      non-missing edges= 20
##
##      Vertex attribute names:
##      priorates totalties vertex.names wealth
##
##      No edge attributes

plot(flomarriage) # Let's view the flomarriage network
```



Remember the general ergm representation of the probability of the observed network, and the conditional log-odds of a tie:

$$\Pr(Y = y) = \exp[\theta'g(y)]/k(\theta)$$

$Y$  is a network;  $g(y)$  is a vector of network stats;  $\theta$  is the vector of coefficients;  $k(\theta)$  is a normalizing constant.

$$\text{logit}(\Pr(Y_{ij} = 1|Y^c)) = \theta' \Delta(g(y))_{ij}$$

$Y_{ij}$  is an actor pair in  $Y$ ;  $Y^c$  is the rest of the network;  $\Delta(g(y))_{ij}$  is the change in  $g(y)$  when the value of  $Y_{ij}$  is toggled on.

We begin with the simplest possible model, the Bernoulli or Erdős-Rényi model, which contains only an edge term.

```
flomodel.01 <- ergm(flomarriage~edges) # fit model

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

flomodel.01

##
## MLE Coefficients:
## edges
## -1.609

summary(flomodel.01) # look in more depth

##
## =====
## Summary of model fit
## =====
##
## Formula:    flomarriage ~ edges
##
## Iterations: 5 out of 30
##
## Monte Carlo MLE Results:
##      Estimate Std. Error MCMC % p-value
## edges  -1.6094      0.2449      0 <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.)
```

How to interpret this model? The log-odds of any tie occurring is:

$$\begin{aligned} & -1.609 \times \text{change in the number of ties} \\ & = -1.609 \times 1 \end{aligned}$$

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

Corresponding probability is:

$$\begin{aligned} & \exp(-1.609)/(1 + \exp(-1.609)) \\ & = 0.1667 \end{aligned}$$

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

Let's add a term often thought to be a measure of "clustering": the number of completed triangles. Note we're in stochastic simulation now – your output will differ

```
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 30:
## Optimizing with step length 1.
## The log-likelihood improved by 0.0426.
## Convergence test p-value: 0.954586084390195. Not converged with 99% confidence; increasing
sample size.
## Iteration 2 of at most 30:
## Optimizing with step length 1.
## The log-likelihood improved by 0.021.
## Convergence test p-value: 0.304573927367486. Not converged with 99% confidence; increasing
sample size.
## Iteration 3 of at most 30:
## Optimizing with step length 1.
## The log-likelihood improved by 0.02931.
## Convergence test p-value: 0.623135250954654. Not converged with 99% confidence; increasing
sample size.
## Iteration 4 of at most 30:
## Optimizing with step length 1.
## The log-likelihood improved by 0.01008.
## Convergence test p-value: 0.0012665338922225. Converged with 99% confidence.
## 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 17 18 19 20 .
## This model was fit using MCMC. To examine model diagnostics and check for degeneracy,
use the mcmc.diagnostics() function.

summary(flomodel.02)

##
## =====
## Summary of model fit
## =====
##
## Formula: flomarriage ~ edges + triangle
##
## Iterations: 4 out of 30
##
## Monte Carlo MLE Results:
## Estimate Std. Error MCMC % p-value
## edges -1.6853 0.3536 0 <1e-04 ***
```

```
## triangle    0.1498    0.6126    0    0.807
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 166.4  on 120  degrees of freedom
## Residual Deviance: 108.0  on 118  degrees of freedom
##
## AIC: 112    BIC: 117.6    (Smaller is better.)
```

```
coef1 = flomodel.02$coef[1]
coef2 = flomodel.02$coef[2]
logodds = coef1 + c(0,1,2) * coef2
expit = function(x) 1/(1+exp(-x))
ps = expit(logodds)
coef1 = round(coef1, 3)
coef2 = round(coef2, 3)
logodds = round(logodds, 3)
ps = round(ps, 3)
```

Again, how to interpret coefficients?

Conditional log-odds of two actors forming a tie is:

$$-1.685 \times \text{change in the number of ties} + 0.15 \times \text{change in number of triangles}$$

- if the tie will not add any triangles to the network, its log-odds is:  $-1.685$ .
- if it will add one triangle to the network, its log-odds is:  $-1.685 + 0.15 = -1.535$
- if it will add two triangles to the network, its log-odds is:  $-1.685 + 0.15 \times 2 = -1.386$
- the corresponding probabilities are 0.156, 0.177, and 0.2.

Let's take a closer look at the ergm object itself:

```
class(flomodel.02) # this has the class ergm

## [1] "ergm"

names(flomodel.02) # let's look straight at the ERGM obj.

## [1] "coef"          "sample"        "sample.obs"
## [4] "iterations"    "MCMCtheta"     "loglikelihood"
## [7] "gradient"      "hessian"       "covar"
## [10] "failure"       "network"       "newnetworks"
## [13] "newnetwork"    "coef.init"     "est.cov"
## [16] "coef.hist"     "stats.hist"    "steplen.hist"
## [19] "control"       "etamap"        "formula"
## [22] "nw.stats"      "constrained"   "constraints"
## [25] "obs.constraints" "reference"     "estimate"
## [28] "offset"        "drop"          "estimable"
## [31] "null.lik"      "mle.lik"
```

```

flomodel.02$coef

##      edges   triangle
## -1.6852567  0.1498013

flomodel.02$formula

## flomarriage ~ edges + triangle

flomodel.02$mle.lik

## 'log Lik.' -54.01907 (df=2)

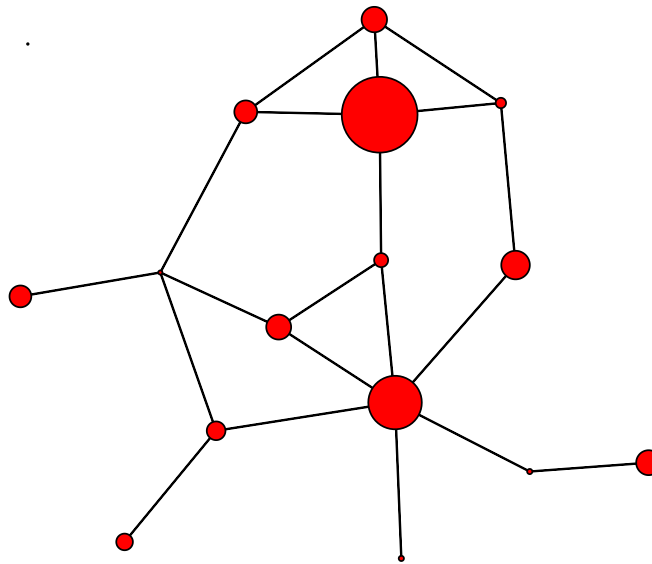
wealth <- flomarriage %v% 'wealth' # the %v% extracts vertex
wealth # attributes from a network

## [1] 10 36 55 44 20 32 8 42 103 48 49 3 27 10 146 48

plot(flomarriage, vertex.cex=wealth/25) # network plot with vertex size

```





*# proportional to wealth*

We can test whether edge probabilities are a function of wealth:

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

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

summary(flomodel.03)

##
## =====
## Summary of model fit
## =====
```

```
##
## Formula:    flomarriage ~ edges + nodecov("wealth")
##
## Iterations: 4 out of 30
##
## Monte Carlo MLE Results:
##              Estimate Std. Error MCMC % p-value
## edges         -2.594929   0.536056      0 <1e-04 ***
## nodecov.wealth  0.010546   0.004674      0 0.0259 *
## ---
## 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.)
```

Yes, there is a significant positive wealth effect on the probability of a tie.

---

Let's try a model or two on:

Is there a statistically significant tendency for ties to be reciprocated ('mutuality')?

```
data(samplk)
ls() # directed data: Sampson's Monks
samplk3
plot(samplk3)
samplmodel.01 <- ergm(samplk3~edges+mutual)
summary(samplmodel.01)
```

Let's try a larger network

```
data(faux.mesa.high)
mesa <- faux.mesa.high
```

```
plot(mesa)
mesa
plot(mesa, vertex.col='Grade')
legend('bottomleft',fill=7:12,legend=paste('Grade',7:12),cex=0.75)
fauxmodel.01 <- ergm(mesa ~edges + nodematch('Grade',diff=T) + nodematch('Race',diff=T))
summary(fauxmodel.01)
```

Note that two of the coefficients are estimated as -Inf (the nodematch coefficients for race Black and Other). Why is this?

```
table(mesa %v% 'Race') # Frequencies of 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

```

So the problem is that there are very few students in the Black and Other race categories, and these students form no homophilous (within-group) ties. The empty cells are what produce the -Inf estimates.

Time to consider some missing data:

```

missnet <- network.initialize(10,directed=F)
missnet[1,2] <- missnet[2,7] <- missnet[3,6] <- 1
missnet[4,6] <- missnet[4,9] <- NA
missnet
plot(missnet)
ergm(missnet~edges)

```

The coefficient equals -2.590. This is the log-odds of the probability .0698. Our network has 3 ties, out of the 43 nodal pairs (10 choose 2 minus 2) whose dyad status we have observed.  $3/43 = 0.0698$ .

```

ergm(missnet~edges+degree(2))
missnet[4,6] <- missnet[4,9] <- 0
ergm(missnet~edges+degree(2))

```

The two estimates for the degree 2 coefficient differ considerably. In the first case, there is one node we know for sure has degree 2, two that may or may not, and seven that we know for sure do not. In the latter, there is one node that has degree 2, and nine that do not.

### 3 Model terms available for *ergm* estimation and simulation

Model terms are the expressions (e.g. “triangle”) used to represent predictors on the right-hand side of equations used in:

- calls to **ergm** (to estimate an ergm model)
- calls to **simulate** (to simulate networks from an ergm model fit)
- calls to **summary** (to obtain measurements of network statistics on a dataset)

#### 3.1 Terms provided with ergm

For a list of available terms that can be used to specify an ERGM, see Appendix B, or type:

```
help('ergm-terms')
```

For a more complete discussion of these terms see the ‘Specifications’ paper in J Stat Software v. 24. (link is available online at [www.statnet.org](http://www.statnet.org))

### 3.2 Coding new terms

We have recently released a new package (`ergm.userterms`) and tutorial aimed at making it much easier than before to write one's own terms. The package is available on CRAN, and installing it will also download the tutorial (`ergmuserterms.pdf`). We teach a workshop at the Sunbelt meetings, and are also hoping for the tutorial to appear soon in the *Journal of Statistical Software*. Note that writing up new `ergm` terms requires some knowledge of C and the ability to build R from source (although the latter is covered in the tutorial).

## 4 Network simulation: the *simulate* command and *network.list* objects

Once we have estimated the coefficients of an ERGM, 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:

```
flomodel.03.sim <- simulate(flomodel.03, nsim=10)
class(flomodel.03.sim)

## [1] "network.list"

summary(flomodel.03.sim)

## Number of Networks: 10
## Model: flomarriage ~ edges + nodecov("wealth")
## Reference: ~Bernoulli
## Constraints: TNT NULL 16      , 1      , 15      , 2      , 14      , 3      , 13      , 4      , 12      , 5      , 11      ,
## Parameters:
##           edges nodecov.wealth
##      -2.59492903      0.01054591
##
## Stored network statistics:
##           edges nodecov.wealth
## [1,]      16      1595
## [2,]      30      3501
## [3,]      19      2459
## [4,]      17      1774
## [5,]      17      1770
## [6,]      23      2504
## [7,]      18      2068
## [8,]      20      1830
## [9,]      17      1861
## [10,]     23      2499
## Number of Networks: 10
## Model: flomarriage ~ edges + nodecov("wealth")
## Reference: ~Bernoulli
## Constraints: TNT NULL 16      , 1      , 15      , 2      , 14      , 3      , 13      , 4      , 12      , 5      , 11      ,
## Parameters:
##           edges nodecov.wealth
##      -2.59492903      0.01054591
```

```

length(flomodel.03.sim)

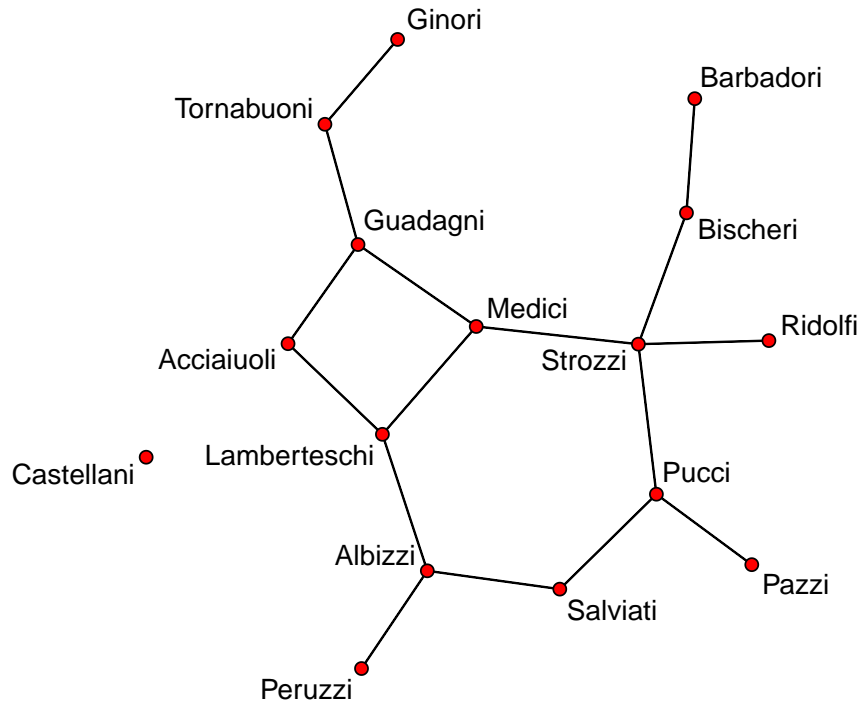
## [1] 10

flomodel.03.sim[[1]]

## Network attributes:
##   vertices = 16
##   directed = FALSE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 16
##   missing edges= 0
##   non-missing edges= 16
##
## Vertex attribute names:
##   priorates totalties vertex.names wealth
##
## No edge attributes

plot(flomodel.03.sim[[1]], label= flomodel.03.sim[[1]] %v% "vertex.names")

```



Voilà. Of course, yours will look somewhat different.

## 5 Examining the quality of model 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 aggregate up to produce characteristic global network properties, even though these global properties are not explicit terms in the model. One test of whether a model “fits the data” is therefore 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.

```

flomodel.03.gof <- gof(flomodel.03~degree)

flomodel.03.gof

##
## Goodness-of-fit for degree
##
##   obs min mean max MC p-value

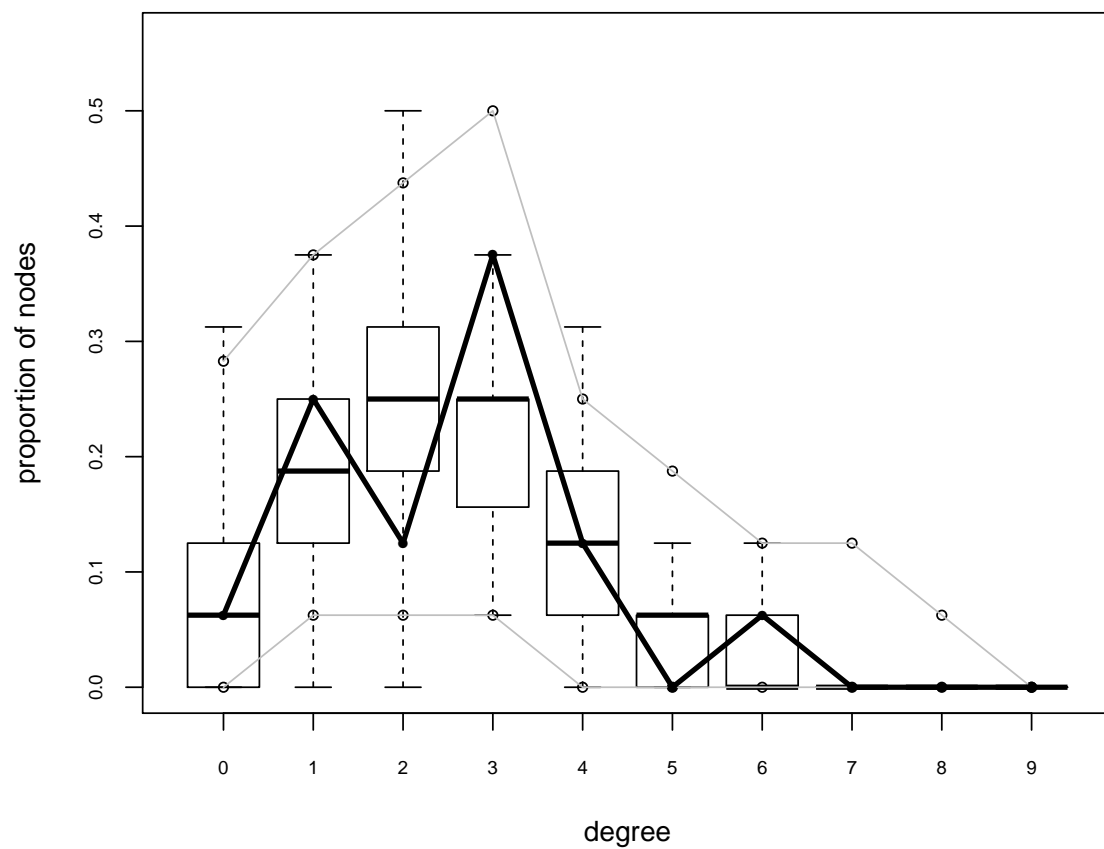
```

```

## 0    1    0 1.34    6    1.00
## 1    4    0 3.40    9    0.92
## 2    2    0 4.12    8    0.28
## 3    6    0 3.64    8    0.22
## 4    2    0 1.95    5    1.00
## 5    0    0 0.78    4    0.92
## 6    1    0 0.40    3    0.66
## 7    0    0 0.22    3    1.00
## 8    0    0 0.12    1    1.00
## 9    0    0 0.01    1    1.00
## 10   0    0 0.01    1    1.00
## 11   0    0 0.01    1    1.00
##
## Goodness-of-fit for model statistics
##
##               obs   min    mean  max MC p-value
## edges                20   12   19.73   29      1.00
## nodecov.wealth 2168 1182 2148.84 3372      0.98

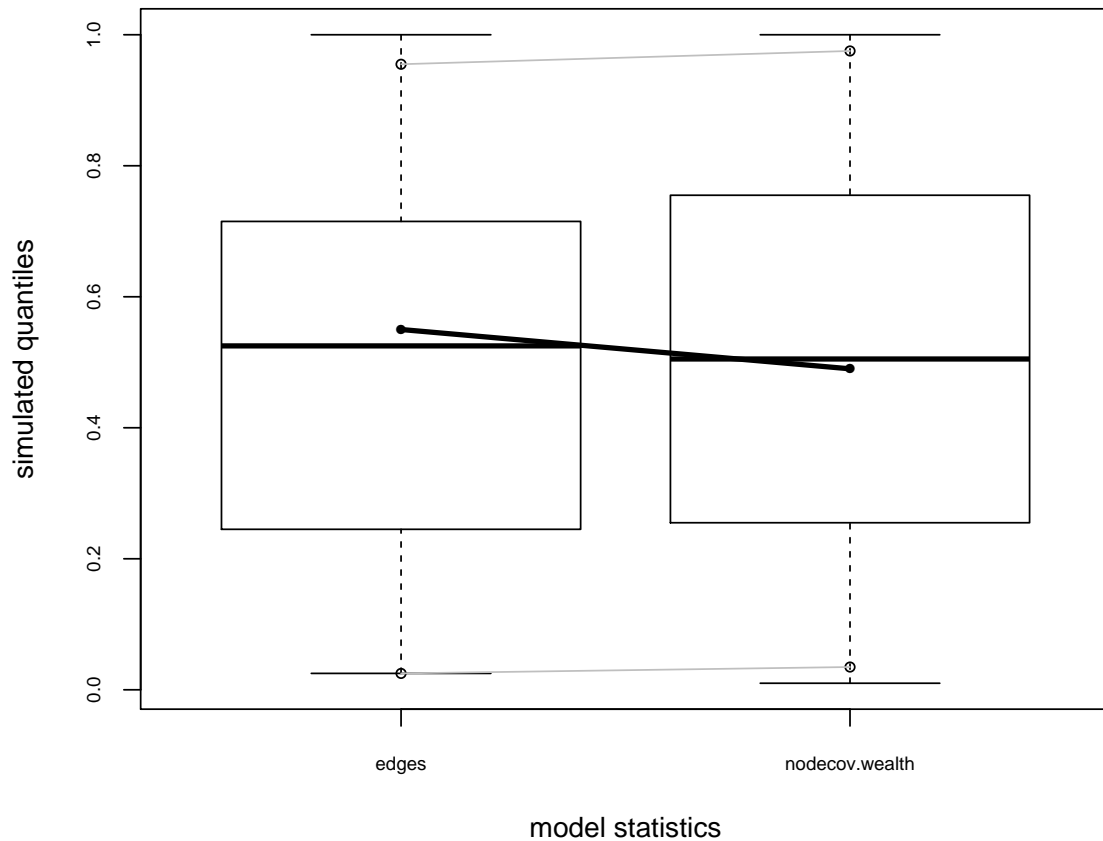
plot(flomodel.03.gof)

```





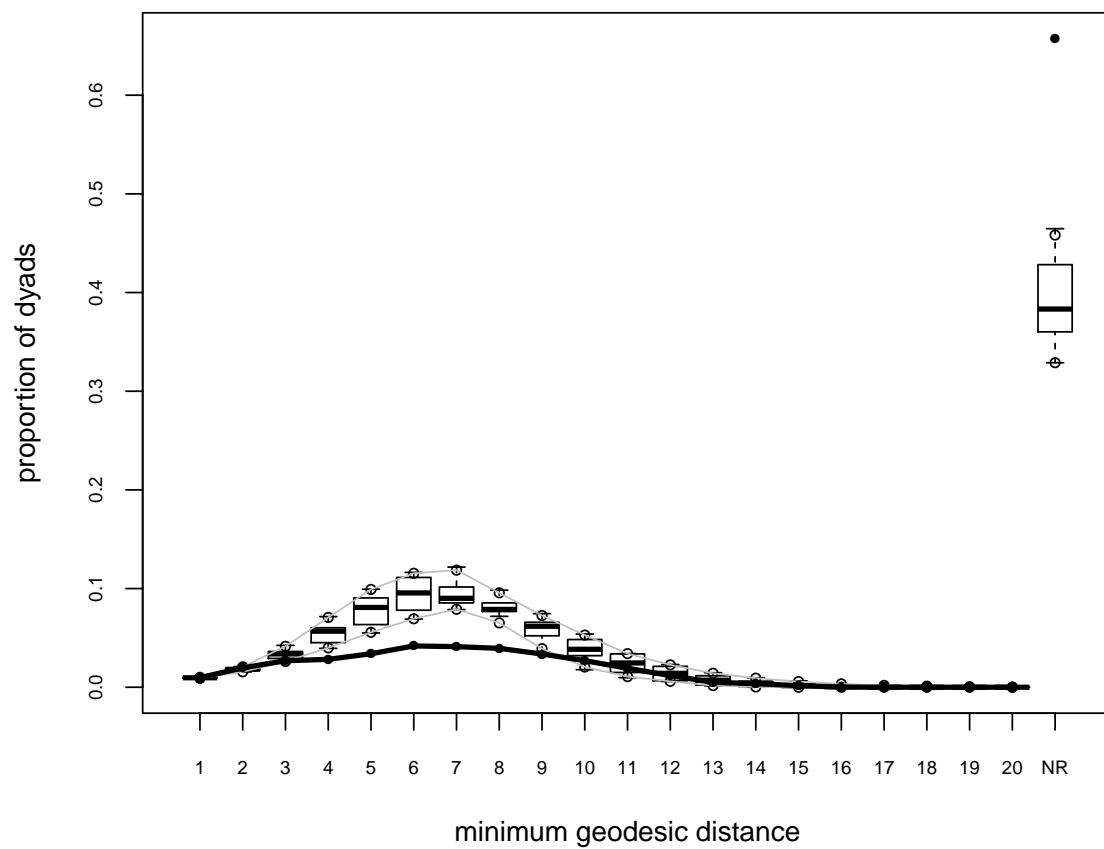
## Goodness-of-fit diagnostics



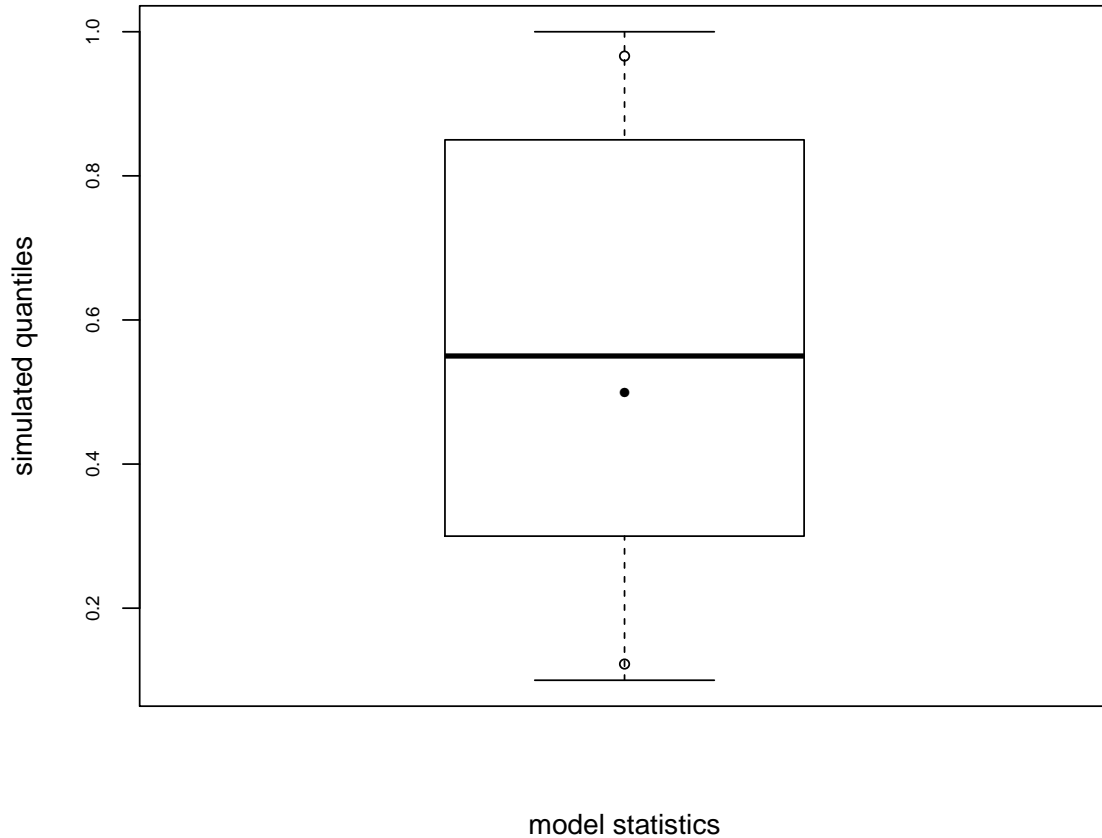
```
mesamodel.02 <- ergm(mesa~edges)

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

mesamodel.02.gof <- gof(mesamodel.02~distance,control=control.gof.ergm(nsim=10))
plot(mesamodel.02.gof)
```



## Goodness-of-fit diagnostics



For a good example of model exploration and fitting for the Add Health Friendship networks, see Goodreau, Kitts & Morris, *Demography* 2009.

## 6 Diagnostics: troubleshooting and checking for model degeneracy

The computational algorithms in `ergm` use 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 simulation 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 more detailed discussions of model degeneracy in the ERGM context, see the papers in *J Stat Software* v. 24. (link is available online at [www.statnet.org](http://www.statnet.org))

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

We will first consider a simulation where the algorithm works. To understand the algorithm, consider

```
fit <- ergm(flobusiness~edges+degree(1),  
  control=control.ergm(MCMC.interval=1, MCMC.burnin=1000, seed=1))
```

This runs a version with every network returned. Let us look at the diagnostics produced:

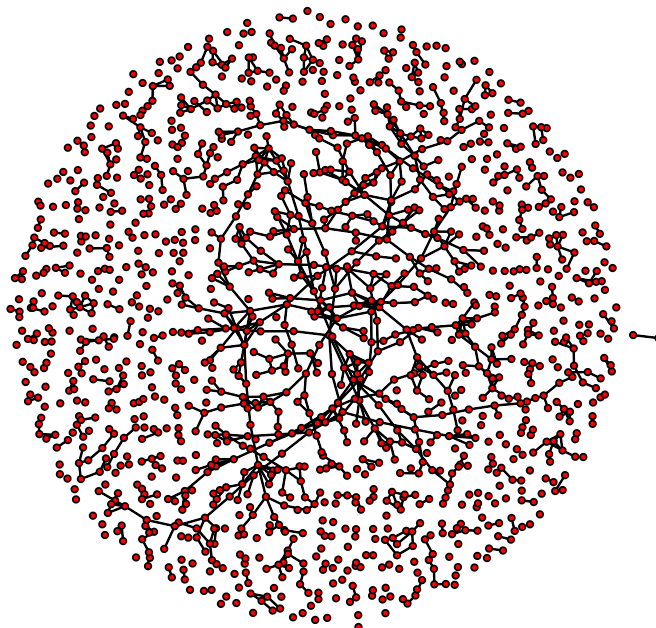
```
mcmc.diagnostics(fit, center=F)
```

Let's look more carefully at a default model fit:

```
fit <- ergm(flobusiness~edges+degree(1))  
mcmc.diagnostics(fit, center=F)
```

Now let us look at a more interesting case, using a larger network:

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



```
fit <- ergm(magnolia~edges+triangle, control=control.ergm(seed=1))
mcmc.diagnostics(fit, center=F)
```

```
## 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 30:
```

```
## Warning in ergm.getMCMCsample(nws, model, MHproposal, mcmc.eta0, control, : Unable  
to reach target effective size in iterations allotted.
```

```
## 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.
```

Very interesting. This model produced degenerate networks. You could have gotten some more feedback about this during the fitting, by using:

```
fit <- ergm(magnolia~edges+triangle, control=control.ergm(seed=1), verbose=T)
```

You might try to increase the MCMC sample size:

```
fit <- ergm(magnolia~edges+triangle,seed=1,
  control = control.ergm(seed=1, MCMC.samplesize=20000),
  verbose=T)
mcmc.diagnostics(fit, center=F)
```

How about trying the more robust version of modeling triangles: GWESP? (For a technical introduction to GWESP see Hunter and Handcock; for a more intuitive description and empirical application see Goodreau Kitts and Morris 2009)

```
fit <- ergm(magnolia~edges+gwesp(0.5,fixed=T),
  control = control.ergm(seed=1, MCML.E.MCMC.precision=0.5))
mcmc.diagnostics(fit)
```

Still degenerate, but maybe getting closer?

```
fit <- ergm(magnolia~edges+gwesp(0.5,fixed=T)+nodematch('Grade')+nodematch('Race')+
  nodematch('Sex'),
  control = control.ergm(seed=1, MCML.E.MCMC.precision=0.5),
  verbose=T)

pdf('diagnostics1.pdf') #Use the recording function if possible, otherwise send to pdf
mcmc.diagnostics(fit)
dev.off() #If you saved to pdf, look at the file

fit <- ergm(magnolia~edges+gwesp(0.25,fixed=T)+nodematch('Grade')+nodematch('Race')+
  nodematch('Sex'),
  control = control.ergm(seed=1, MCML.E.MCMC.precision=0.5))
mcmc.diagnostics(fit)
```

One more try...

```
fit <- ergm(magnolia~edges+gwesp(0.25,fixed=T)+nodematch('Grade')+nodematch('Race')+
  nodematch('Sex'),
  control = control.ergm(seed=1, MCML.E.MCMC.precision=0.5),
  verbose=T)

## Evaluating network in model.
## Initializing Metropolis-Hastings proposal(s): Unconstrained ergm:MH_TNT
## Initializing model.
## Using initial method 'MPLE'.
## Fitting initial model.
## Starting maximum pseudolikelihood estimation (MPLE):
## Evaluating the predictor and response matrix.
## MPLE covariate matrix has 211 rows.
## 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 30 with parameter:
##           edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      -9.8619687          1.6946112          2.8534613          0.9886313
##      nodematch.Sex
##           0.8245285
## Beginning adaptive MCMC...
## First run: running each chain forward by 640 steps with interval 32.
## No adequate burn-in found. Best convergence p-value = 0.0261240961553919
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 640 steps with interval 32.
## ESS of 11.0950917210067 attained with burn-in of 12.5%; convergence p-value = 0.809804572448642.
## Predicted additional sample size: 5340.51441506208 dampened to 3112.2641295318, so
running 3112 steps forward.
## Increasing thinning to 64.
## Increasing thinning to 128.
## No adequate burn-in found. Best convergence p-value = 0.00677879016342437
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 640 steps with interval 128.
## No adequate burn-in found. Best convergence p-value = 0.185611451396309
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 640 steps with interval 128.
## Increasing thinning to 256.
## ESS of 20.9070531805928 attained with burn-in of 49.96%; convergence p-value = 0.229989631209895.
## Predicted additional sample size: 1226.39489822259 dampened to 1036.66639360061, so
running 1037 steps forward.
## Increasing thinning to 512.
## ESS of 31.2542426025999 attained with burn-in of 49.96%; convergence p-value = 0.582013927375938.
## Predicted additional sample size: 583.581150958192 dampened to 597.257041117002, so
running 597 steps forward.
## No adequate burn-in found. Best convergence p-value = 0.114362978404196
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 640 steps with interval 512.
## Increasing thinning to 1024.
## No adequate burn-in found. Best convergence p-value = 0.0444700029549334
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 640 steps with interval 1024.
## Increasing thinning to 2048.
## No adequate burn-in found. Best convergence p-value = 0.00343160234840296
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 640 steps with interval 2048.
## ESS of 125.10344062942 attained with burn-in of 25.02%; convergence p-value = 0.34163987038281.
## Target ESS achieved and is trustworthy. Returning.
## Sample size = 1160 by 640.
## Back from unconstrained MCMC.
## New interval = 1024.
## Average estimating function values:
##           edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      -99.82672          -39.58344          -97.15862          -92.88879

```

```

##      nodematch.Sex
##      -82.23017
## Starting MCMLE Optimization...
## Optimizing with step length 0.759305672892666.
## Using lognormal metric (see control.ergm function).
## Using log-normal approx (no optim)
## The log-likelihood improved by 3.115.
## Estimating equations are not within tolerance region.
##
## Iteration 2 of at most 30 with parameter:
##      edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      -9.8034540      1.7438535      2.7697852      0.9462849
##      nodematch.Sex
##      0.7767260
## Beginning adaptive MCMC...
## First run: running each chain forward by 640 steps with interval 1024.
## No adequate burn-in found. Best convergence p-value = 0.0576355859406755
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
## remaining: running forward by 640 steps with interval 1024.
## ESS of 82.0493775745156 attained with burn-in of 6.25%; convergence p-value = 0.264626815912483.
## Target ESS achieved and is trustworthy. Returning.
## Sample size = 1200 by 640.
## Back from unconstrained MCMC.
## New interval = 512.
## Average estimating function values:
##      edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      -40.83667      -27.00118      -39.02667      -47.32833
##      nodematch.Sex
##      -37.59917
## 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 1.821.
## Distance from origin on tolerance region scale: 7.27992662864131 (previously 24.4839522713924).
## Estimating equations are not within tolerance region.
##
## Iteration 3 of at most 30 with parameter:
##      edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      -9.6875406      1.7286120      2.7441523      0.8369131
##      nodematch.Sex
##      0.7291659
## Beginning adaptive MCMC...
## First run: running each chain forward by 640 steps with interval 512.
## No adequate burn-in found. Best convergence p-value = 0.170860982349133
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
## remaining: running forward by 640 steps with interval 512.
## No adequate burn-in found. Best convergence p-value = 0.0516563547523613
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
## remaining: running forward by 640 steps with interval 512.
## Increasing thinning to 1024.

```



```

## No adequate burn-in found. Best convergence p-value = 0.103298955306552
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 640 steps with interval 1024.
## No adequate burn-in found. Best convergence p-value = 6.13553533849147e-09
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 640 steps with interval 1024.
## Increasing thinning to 2048.
## No adequate burn-in found. Best convergence p-value = 7.06626033050952e-05
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 640 steps with interval 2048.
## Increasing thinning to 4096.
## No adequate burn-in found. Best convergence p-value = 0.00642396256925009
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 640 steps with interval 4096.
## Increasing thinning to 8192.
## No adequate burn-in found. Best convergence p-value = 0.000762795674173389
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 640 steps with interval 8192.
## No adequate burn-in found. Best convergence p-value = 0.000102568750785848
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 640 steps with interval 8192.
## Increasing thinning to 16384.
## ESS of 183.393318440151 attained with burn-in of 3.14%; convergence p-value = 0.207805372313508.
## Target ESS achieved and is trustworthy. Returning.
## Sample size = 988 by 640.
## Back from unconstrained MCMC.
## New interval = 8192.
## Average estimating function values:
##           edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##           80.01822           92.74033           80.60324           84.65587
##           nodematch.Sex
##           68.64170
## Starting MCMLL Optimization...
## Optimizing with step length 0.952847627362893.
## Using lognormal metric (see control.ergm function).
## Using log-normal approx (no optim)
## The log-likelihood improved by 3.649.
## Distance from origin on tolerance region scale: 16.0581472802508 (previously 4.33524375618965).
## Estimating equations are not within tolerance region.
##
## Iteration 4 of at most 30 with parameter:
##           edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##           -9.7564131           1.8076049           2.7441039           0.8863056
##           nodematch.Sex
##           0.7534368
## Beginning adaptive MCMC...
## First run: running each chain forward by 640 steps with interval 8192.
## No adequate burn-in found. Best convergence p-value = 0.00408405305548725
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 640 steps with interval 8192.
## No adequate burn-in found. Best convergence p-value = 0.0709921506031077

```

```

## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 640 steps with interval 8192.
## Increasing thinning to 16384.
## ESS of 169.654825301385 attained with burn-in of 12.5%; convergence p-value = 0.201533573302803.
## Target ESS achieved and is trustworthy. Returning.
## Sample size = 840 by 640.
## Back from unconstrained MCMC.
## New interval = 8192.
## Average estimating function values:
##           edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      -11.733333      -13.742112      -12.164286      -5.458333
##      nodematch.Sex
##      -10.055952
## Starting MCML 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 0.1346.
## Distance from origin on tolerance region scale: 0.537852694366175 (previously 18.2384664308051).
## Test statistic:  $T^2 = 0.812858786947969$ , with 5 free parameters and 168.654825301385
degrees of freedom.
## Convergence test p-value: 0.977106140177512. Not converged with 99% confidence; increasing
sample size.
## 99% confidence critical value = 16.031662728387.
##
## Iteration 5 of at most 30 with parameter:
##           edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      -9.7657417      1.7953952      2.7370260      0.9117911
##      nodematch.Sex
##      0.7489603
## Beginning adaptive MCMC...
## First run: running each chain forward by 905 steps with interval 8192.
## ESS of 144.050652711352 attained with burn-in of 24.97%; convergence p-value = 0.793038822335618.
## Target ESS achieved and is trustworthy. Returning.
## Sample size = 679 by 905.
## Back from unconstrained MCMC.
## New interval = 4096.
## Average estimating function values:
##           edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      24.91016      22.41496      24.46834      19.98822
##      nodematch.Sex
##      20.31811
## Starting MCML 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 0.4815.
## Distance from origin on tolerance region scale: 1.92335685410117 (previously 1.05041407793716).
## Estimating equations are not within tolerance region.

```

```

## Estimating equations did not move closer to tolerance region more than 1 time(s) in
4 steps; increasing sample size.
##
## Iteration 6 of at most 30 with parameter:
##          edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##          -9.7819432          1.8243240          2.7498820          0.8988278
##          nodematch.Sex
##          0.7815809
## Beginning adaptive MCMC...
## First run: running each chain forward by 1280 steps with interval 4096.
## No adequate burn-in found. Best convergence p-value = 0.0279062224020801
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 1280 steps with interval 4096.
## ESS of 175.769989278553 attained with burn-in of 12.5%; convergence p-value = 0.776065331579211.
## Predicted additional sample size: 1022.44543993934 dampened to 1036.30960395533, so
running 1036 steps forward.
## Increasing thinning to 8192.
## No adequate burn-in found. Best convergence p-value = 0.177363110158637
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 1280 steps with interval 8192.
## ESS of 286.677281541888 attained with burn-in of 6.24%; convergence p-value = 0.807883548232271.
## Target ESS achieved and is trustworthy. Returning.
## Sample size = 2886 by 1280.
## Back from unconstrained MCMC.
## New interval = 4096.
## Average estimating function values:
##          edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##          -45.91026          -45.71503          -44.65835          -39.50416
##          nodematch.Sex
##          -37.02599
## Starting MCML E 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 0.4011.
## Distance from origin on tolerance region scale: 1.60395107697489 (previously 0.488884895373582).
## Estimating equations are not within tolerance region.
##
## Iteration 7 of at most 30 with parameter:
##          edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##          -9.7792032          1.8154126          2.7455184          0.9023243
##          nodematch.Sex
##          0.7690515
## Beginning adaptive MCMC...
## First run: running each chain forward by 1280 steps with interval 4096.
## ESS of 140.891532446786 attained with burn-in of 25%; convergence p-value = 0.77686268321089.
## Predicted additional sample size: 784.320582876914 dampened to 817.170631457201, so
running 817 steps forward.
## ESS of 228.893090624637 attained with burn-in of 12.49%; convergence p-value = 0.29431423710472.
## Predicted additional sample size: 217.311840073683 dampened to 261.795678310598, so

```

```

running 262 steps forward.
## No adequate burn-in found. Best convergence p-value = 0.0145950164138075
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 1280 steps with interval 4096.
## ESS of 271.845724909087 attained with burn-in of 6.24%; convergence p-value = 0.78805257360016.
## Target ESS achieved and is trustworthy. Returning.
## Sample size = 3412 by 1280.
## Back from unconstrained MCMC.
## New interval = 2048.
## Average estimating function values:
##          edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      -22.09555      -24.40488      -21.96512      -19.29601
##      nodematch.Sex
##      -16.68142
## 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 0.2937.
## Distance from origin on tolerance region scale: 1.17428610772358 (previously 4.44957103205913).
## Estimating equations are not within tolerance region.
##
## Iteration 8 of at most 30 with parameter:
##          edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      -9.7596032      1.7921977      2.7425435      0.8929461
##      nodematch.Sex
##      0.7566109
## Beginning adaptive MCMC...
## First run: running each chain forward by 1280 steps with interval 2048.
## No adequate burn-in found. Best convergence p-value = 0.0637250946224051
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 1280 steps with interval 2048.
## ESS of 139.124306637077 attained with burn-in of 50%; convergence p-value = 0.38029502066258.
## Predicted additional sample size: 1075.30374181698 dampened to 1089.03029403366, so
running 1089 steps forward.
## No adequate burn-in found. Best convergence p-value = 0.00580219189021897
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 1280 steps with interval 2048.
## Increasing thinning to 4096.
## ESS of 156.06065468888 attained with burn-in of 25%; convergence p-value = 0.234796187028316.
## Predicted additional sample size: 1183.43672531133 dampened to 1189.25166533246, so
running 1189 steps forward.
## Increasing thinning to 8192.
## ESS of 124.726265433576 attained with burn-in of 50%; convergence p-value = 0.270425340470473.
## Predicted additional sample size: 960.927670226561 dampened to 984.610678788151, so
running 985 steps forward.
## No adequate burn-in found. Best convergence p-value = 0.00248922378636977
## Insufficient ESS or untrustworthy burn-in estimate to determine the number of steps
remaining: running forward by 1280 steps with interval 8192.
## Increasing thinning to 16384.

```

```

## ESS of 277.81101772385 attained with burn-in of 12.52%; convergence p-value = 0.244175894265473.
## Target ESS achieved and is trustworthy. Returning.
## Sample size = 1789 by 1280.
## Back from unconstrained MCMC.
## New interval = 8192.
## Average estimating function values:
##           edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##           24.36836          24.13337          24.67300          25.73896
##           nodematch.Sex
##           23.89603
## Starting MCML E 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 0.2683.
## Distance from origin on tolerance region scale: 1.07272006295203 (previously 0.761465705301606).
## Estimating equations are not within tolerance region.
## Estimating equations did not move closer to tolerance region more than 1 time(s) in
4 steps; increasing sample size.
##
## Iteration 9 of at most 30 with parameter:
##           edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##           -9.7862753          1.7967783          2.7497889          0.9074249
##           nodematch.Sex
##           0.7785660
## Beginning adaptive MCMC...
## First run: running each chain forward by 1810 steps with interval 8192.
## ESS of 187.790048224011 attained with burn-in of 6.24%; convergence p-value = 0.333205980283223.
## Predicted additional sample size: 2929.78405068383 dampened to 2873.16926845699, so
running 2873 steps forward.
## Increasing thinning to 16384.
## ESS of 247.588389699925 attained with burn-in of 49.98%; convergence p-value = 0.242420865190371.
## Predicted additional sample size: 1250.56750858411 dampened to 1272.28558726257, so
running 1272 steps forward.
## ESS of 409.392385332951 attained with burn-in of 24.99%; convergence p-value = 0.22465687941141.
## Predicted additional sample size: 679.217898793001 dampened to 706.180288687354, so
running 706 steps forward.
## ESS of 540.327012128828 attained with burn-in of 6.25%; convergence p-value = 0.371086478242861.
## Target ESS achieved and is trustworthy. Returning.
## Sample size = 4049 by 1810.
## Back from unconstrained MCMC.
## New interval = 8192.
## Average estimating function values:
##           edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##           -10.081008          -8.575304          -9.794023          -8.516177
##           nodematch.Sex
##           -10.323784
## Starting MCML E 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 0.04624.
## Distance from origin on tolerance region scale: 0.184900083229536 (previously 1.15699263252127).
## Test statistic:  $T^2 = 9.36952556028547$ , with 5 free parameters and 539.327012128827
degrees of freedom.
## Convergence test p-value: 0.0996472639791397. Not converged with 99% confidence;
increasing sample size.
## 99% confidence critical value = 15.3716570523432.
##
## Iteration 10 of at most 30 with parameter:
##          edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      -9.7800568          1.7956405          2.7465057          0.9099125
##      nodematch.Sex
##          0.7655441
## Beginning adaptive MCMC...
## First run: running each chain forward by 2318 steps with interval 8192.
## ESS of 249.066420140315 attained with burn-in of 12.51%; convergence p-value = 0.566175096679505.
## Predicted additional sample size: 4811.52307840565 dampened to 4715.2727393206, so
running 4715 steps forward.
## Increasing thinning to 16384.
## ESS of 509.295989549033 attained with burn-in of 3.13%; convergence p-value = 0.756082479025658.
## Predicted additional sample size: 2211.55631168451 dampened to 2213.60608080381, so
running 2214 steps forward.
## Increasing thinning to 32768.
## ESS of 648.236570873325 attained with burn-in of 3.14%; convergence p-value = 0.226286470549855.
## Predicted additional sample size: 820.857777035928 dampened to 843.602520935187, so
running 844 steps forward.
## ESS of 650.383524429103 attained with burn-in of 24.99%; convergence p-value = 0.892857304485958.
## Predicted additional sample size: 811.028332502513 dampened to 833.847976114916, so
running 834 steps forward.
## ESS of 1021.26432045394 attained with burn-in of 3.13%; convergence p-value = 0.302367689599081.
## Target ESS achieved and is trustworthy. Returning.
## Sample size = 4401 by 2318.
## Back from unconstrained MCMC.
## New interval = 16384.
## Average estimating function values:
##          edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      0.8918428          1.9694339          1.3746876          0.2626676
##      nodematch.Sex
##      1.2535787
## Starting MCMLL 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 0.004197.
## Distance from origin on tolerance region scale: 0.0167851301116829 (previously 0.180383022873337)
## Test statistic:  $T^2 = 51.9717387734731$ , with 5 free parameters and 1020.26432045395
degrees of freedom.
## Convergence test p-value: 1.06092250862656e-09. Converged with 99% confidence.

```

```
## 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 17 18 19 20 .
## 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 = 4685824:148865024
```

```
## Thinning interval = 32768
```

```
## Number of chains = 1
```

```
## Sample size per chain = 4401
```

```
##
```

```
## 1. Empirical mean and standard deviation for each variable,
```

```
## plus standard error of the mean:
```

```
##
```

	Mean	SD	Naive SE	Time-series SE
edges	-0.8918	45.24	0.6820	3.347
gwesp.fixed.0.25	-1.9694	40.00	0.6030	3.365
nodematch.Grade	-1.3747	43.02	0.6484	3.321
nodematch.Race	-0.2627	41.40	0.6241	3.222
nodematch.Sex	-1.2536	36.12	0.5444	2.700

```
##
```

```
## 2. Quantiles for each variable:
```

```
##
```

	2.5%	25%	50%	75%	97.5%
edges	-88.00	-32.00	-2.0	31.00	89.00
gwesp.fixed.0.25	-72.46	-31.47	-3.6	25.44	78.44
nodematch.Grade	-84.00	-32.00	-3.0	28.00	83.00
nodematch.Race	-77.00	-29.00	-2.0	28.00	83.00
nodematch.Sex	-69.00	-27.00	-2.0	23.00	72.00

```
##
```

```
##
```

```
## Sample statistics cross-correlations:
```

	edges	gwesp.fixed.0.25	nodematch.Grade	nodematch.Race
edges	1.0000000	0.8380189	0.9582470	0.9452463
gwesp.fixed.0.25	0.8380189	1.0000000	0.8579551	0.8303668
nodematch.Grade	0.9582470	0.8579551	1.0000000	0.9143098
nodematch.Race	0.9452463	0.8303668	0.9143098	1.0000000
nodematch.Sex	0.8975064	0.7732280	0.8645874	0.8462788

```
##
```

```
## nodematch.Sex
```

```
## edges 0.8975064
```

```
## gwesp.fixed.0.25 0.7732280
```

```
## nodematch.Grade 0.8645874
```

```
## nodematch.Race 0.8462788
```

```
## nodematch.Sex 1.0000000
```

```
##
```

```
## Sample statistics auto-correlation:
```

```
## Chain 1
```

```

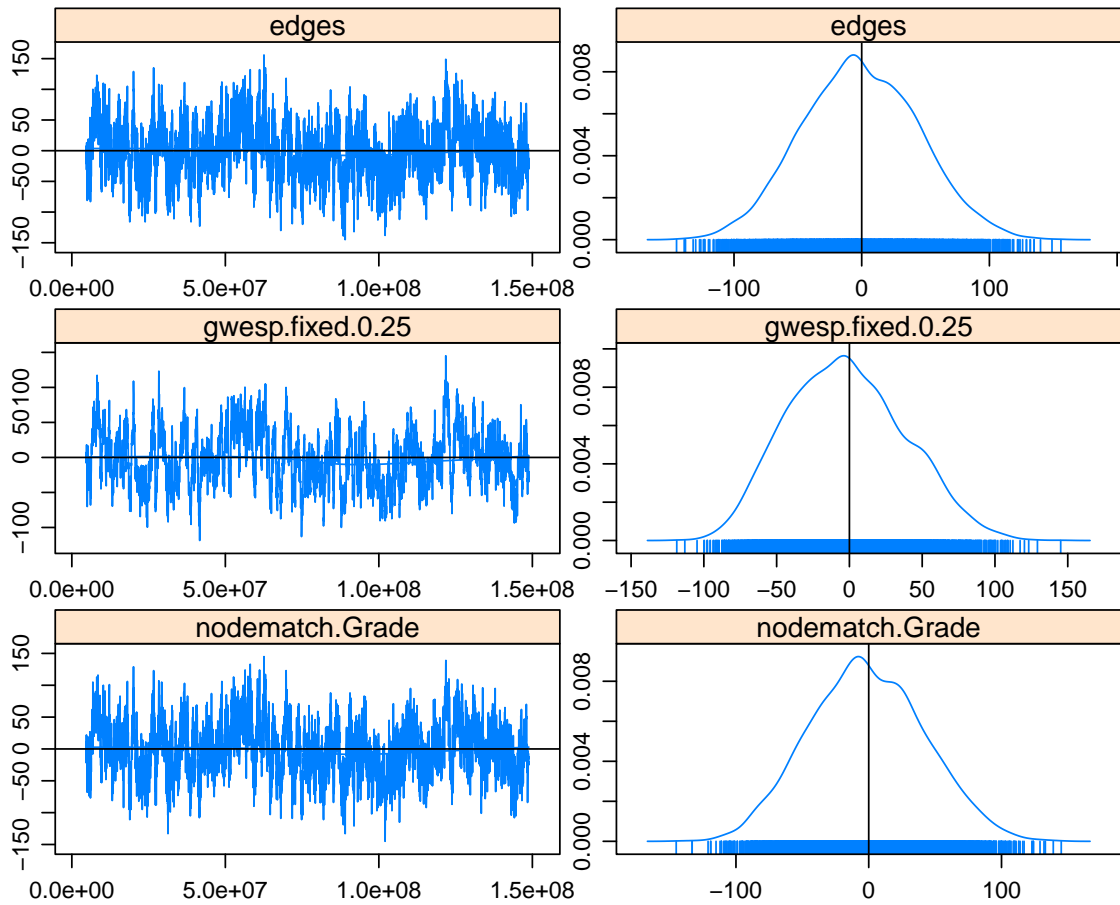
##          edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
## Lag 0      1.0000000      1.0000000      1.0000000      1.0000000
## Lag 32768  0.6662337      0.9301994      0.7172254      0.6971997
## Lag 65536  0.6206129      0.8732211      0.6748981      0.6563421
## Lag 98304  0.5809752      0.8214928      0.6352331      0.6164264
## Lag 131072 0.5394254      0.7754180      0.5960030      0.5744829
## Lag 163840 0.5093723      0.7330603      0.5626690      0.5460681
##          nodematch.Sex
## Lag 0      1.0000000
## Lag 32768  0.6552672
## Lag 65536  0.6172034
## Lag 98304  0.5869156
## Lag 131072 0.5461642
## Lag 163840 0.5240576
##
## 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
##          2.062      1.821      1.874      1.987
##          nodematch.Sex
##          1.721
##
## Individual P-values (lower = worse):
##          edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##          0.03917851      0.06868225      0.06088087      0.04695697
##          nodematch.Sex
##          0.08530083
## Joint P-value (lower = worse): 0.3023677 .

## Warning in formals(fun): argument is not a function

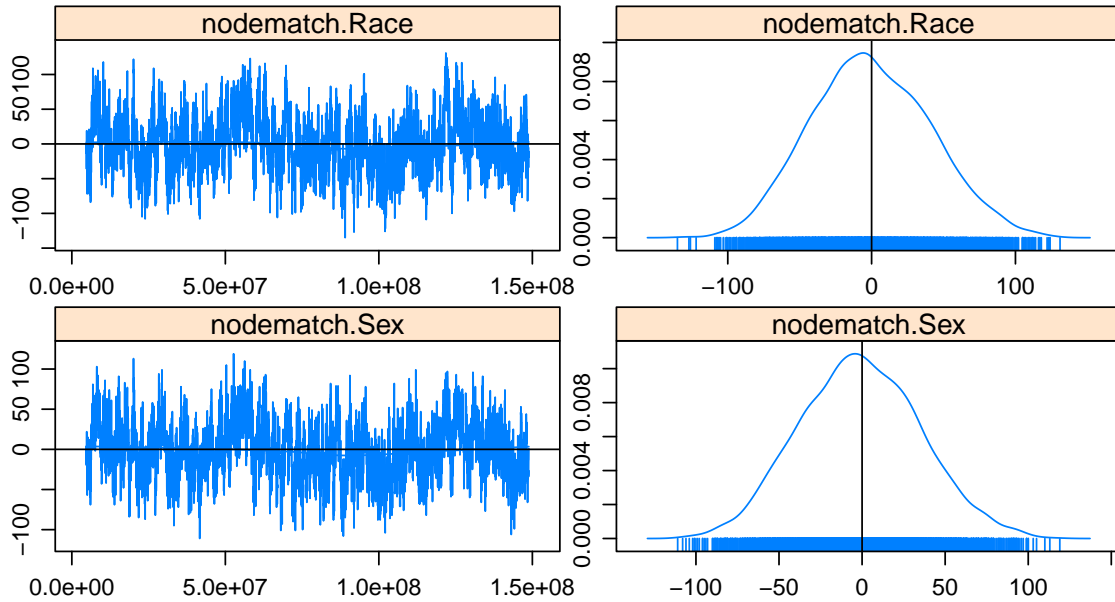
```



## Sample statistics



## Sample statistics



```
##
```

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

Success! Of course, in real life one might have a lot more trial and error.

Changes in version 3.2 of the `ergm` estimation algorithm mean that the MCMC diagnostic plots can no longer be used to ensure that the mean statistics from the model match the observed network statistics. For that functionality, please use the GOF command: `gof(fit, GOF=~model)`. The plots can still be used to assess MCMC mixing and convergence.

## 7 Working with egocentrically sampled network data

In many empirical contexts, it is not feasible to collect a network census or even an adaptive (link-traced) sample. Even when one of these may be possible in practice, egocentrically sampled data are typically cheaper and easier to collect.

Long regarded as the poor country cousin in the network data family, egocentric data contain a remarkable amount of information. With the right statistical methods, such data can be used to explore the properties of the complete networks in which they are embedded. The basic idea here is to

combine what is observed, with assumptions, to define a class of models that describe the distribution of networks that are centered on the observed properties. The variation in these networks quantifies some of the uncertainty introduced by the assumptions.

Let’s start with a simple fictional example: You have a sample of persons who were asked about the friends they had seen face-to-face more than once in the last week. The respondent was asked their own sex, and the sex of each friend (for up to three friends). Summary statistics from these data thus include the sex distribution, the degree distribution (it could be broken down by sex, but we will just examine the marginal distribution here), and the joint distribution of the respondent and friend’s sex (the sex mixing matrix). Let’s assume there are equal numbers of men and women in the sampled respondents. The other distributions are shown below:

Degree distribution				Sex mixing matrix			
Degree	Frequency	Fraction	Ties	(410 total)	Friend		
0	180	0.36	0	<b>Respondent</b>		M	F
1	245	0.49	245				
2	60	0.12	120		M	164	44
3	15	0.03	45		F	26	176
Total	500	1.00	410				

So, total N respondents = 500, total N friends reported = 410.

We can use an ERGM to fit the parameters associated with these observed statistics, then use the fitted model to simulate complete networks that are drawn from the distribution of networks that is centered around these statistics. As a theoretical exercise, this provides a method for investigating the complete network implications of these observed summary statistics. As an empirical exercise, the primary assumption needed for inference is that the data we have is sampled from a population in equilibrium (and, as in all statistical inference, that our model is correct). The theory for this is developed in Krivitsky, 2009.

We need to make assumptions about size, directedness and bipartite-ness when we model and simulate the complete network.

- **Size:** any size can be simulated, but if the model is fit using the observed frequencies, it should be used to simulate a population of that size, unless a size adjustment is made in the simulation (see Krivitsky, Handcock and Morris 2011). We are going to work with a population size 500 here, equal to the number of respondents.
- **Directedness:** Ego data are in one sense inherently directed (respondents nominate alters, alters are not observed), but the relationship may be either. In this case (“seen more than once”) it is undirected, so we will fit and simulate an undirected network.
- **Bipartite:** Ego data can be bipartite (if no alters are also respondents, or data are collected on 2-mode networks) or not (if respondents are also alters). But again, the relationship may be either. “Seen” is undirected, and we will fit and simulated and undirected network.

In sum, we will simulate a one-mode, undirected network of size 500, assuming the ego statistics we observed contain the information we need to calculate the statistics that would have been observed in a self-contained population of that size, noting that other assumptions are possible.

To ensure consistency between the degree distribution (which is a tabulation of nodes) and the mixing matrix (which is a cross-tabulation of ties) in our simulated “complete network,” it is important to recognize that in a complete network, the degree distribution should imply twice the number of ties observed in the mixing matrix, because every tie is being reported by both nodes in the degree distribution. If we are fixing the population size at 500 in this simulation, then our observed mixing matrix data needs to be divided by 2.

Start by initializing an empty network of the desired size and assign the “sex” attribute to the nodes:

```
ego.net <- network.initialize(500, directed=F)
ego.net %v% 'sex' <- c(rep(0,250),rep(1,250))
```

Set up the observed statistics (adjusted for this “complete” network) as we will use them to assess the accuracy of the simulation later:

```
ego.deg <- c(180, 245, 60, 15) # node distn
ego.mixmat <- matrix(c(164,44,26,176)/2, nrow=2, byrow=T) # adjusted tie distn
```

Then, pick the observed statistics you want to target – we will start with a simple model here: the total number of ties (edges), and the number of sex-matched ties (homophily). These are both functions of the observed statistics:

```
ego.edges <- sum(ego.mixmat)
ego.sexmatch <- ego.mixmat[1,1]+ego.mixmat[2,2]
```

And combine these into a vector

```
ego.target.stats <- c(ego.edges, ego.sexmatch)
ego.target.stats

## [1] 205 170
```

Now, fit an ERGM to this “network”, with terms for the statistics you want to match, and the “target.stats” argument for **ergm** that specifies the target values for those statistics:

```
ego.fit <- ergm(ego.net ~ edges + nodematch('sex'),
  target.stats = ego.target.stats)

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

Take a look at the fitted model:

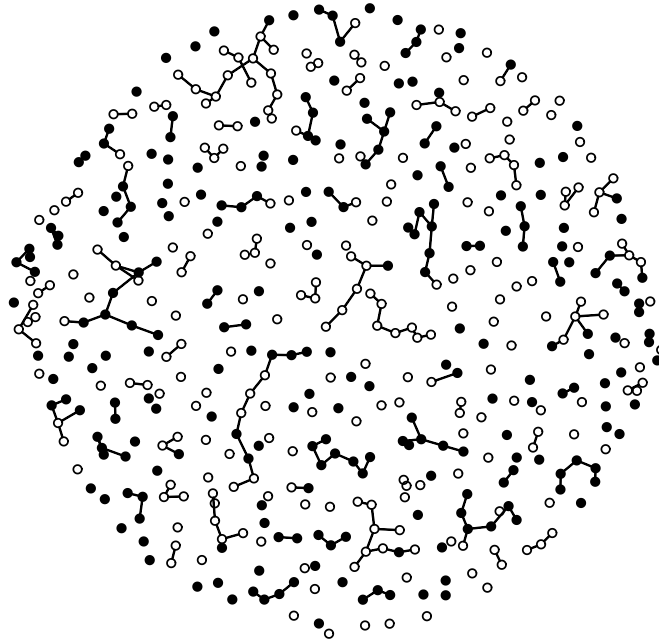
```
summary(ego.fit)

##
## =====
```

```
## Summary of model fit
## =====
##
## Formula:   nw ~ edges + nodematch("sex")
## <environment: 0x55c76c1f8540>
##
## Iterations: 8 out of 30
##
## Monte Carlo MLE Results:
##           Estimate Std. Error MCMC % p-value
## edges      -7.4870    0.1690      0 <1e-04 ***
## nodematch.sex  1.5866    0.1857      0 <1e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 172940 on 124750 degrees of freedom
## Residual Deviance:  2941 on 124748 degrees of freedom
##
## AIC: 2945    BIC: 2964    (Smaller is better.)
```

Now that you have a fitted model, you can simulate a complete network from it, and look at the results:

```
ego.sim1 <- simulate(ego.fit)
plot(ego.sim1, vertex.cex=.65, vertex.col="sex")
```



Does it reproduce the observed degree and mixing frequencies? We only targeted the total number of edges, not the full degree distribution.

```
rbind(summary(ego.sim1 ~ degree(c(0:3))), ego.deg)
```

```
##           degree0 degree1 degree2 degree3
##           222     175     85     17
## ego.deg      180     245     60     15
```

```
mixingmatrix(ego.sim1, "sex")
```

```
## Note: Marginal totals can be misleading
## for undirected mixing matrices.
```

```
##    0  1
## 0 84 29
## 1 29 87
```

```
ego.mixmat
```

```
##      [,1] [,2]
## [1,]  82  22
## [2,]  13  88
```

We only targeted the number of same-sex ties, not the full mixing matrix.

The simulation stats seem quite different than the observed stats, and there are two possible reasons: either we mis-specified the original model (bias), or this one random draw may be unrepresentative of the distribution described by the model (variance). The latter is easily examined by simulating 100 networks, to see where the observed data fall in the distribution of networks produced by the model:

```
ego.sim100 <- simulate(ego.fit, nsim=100)
ego.sim100

## Number of Networks: 100
## Model: nw ~ edges + nodematch("sex")
## Reference: ~Bernoulli
## Constraints: TNT NULL 500 , 1 , 499 , 2 , 498 , 3 , 497 , 4 , 496 , 5 , 495 ,
## Parameters:
##          edges nodematch.sex
##      -7.487013      1.586633
```

More information can be obtained with

```
summary(ego.sim100)
```

First, we'll look at how well the simulations reproduced the target statistics that were included in the model (note, not the observed full distributions):

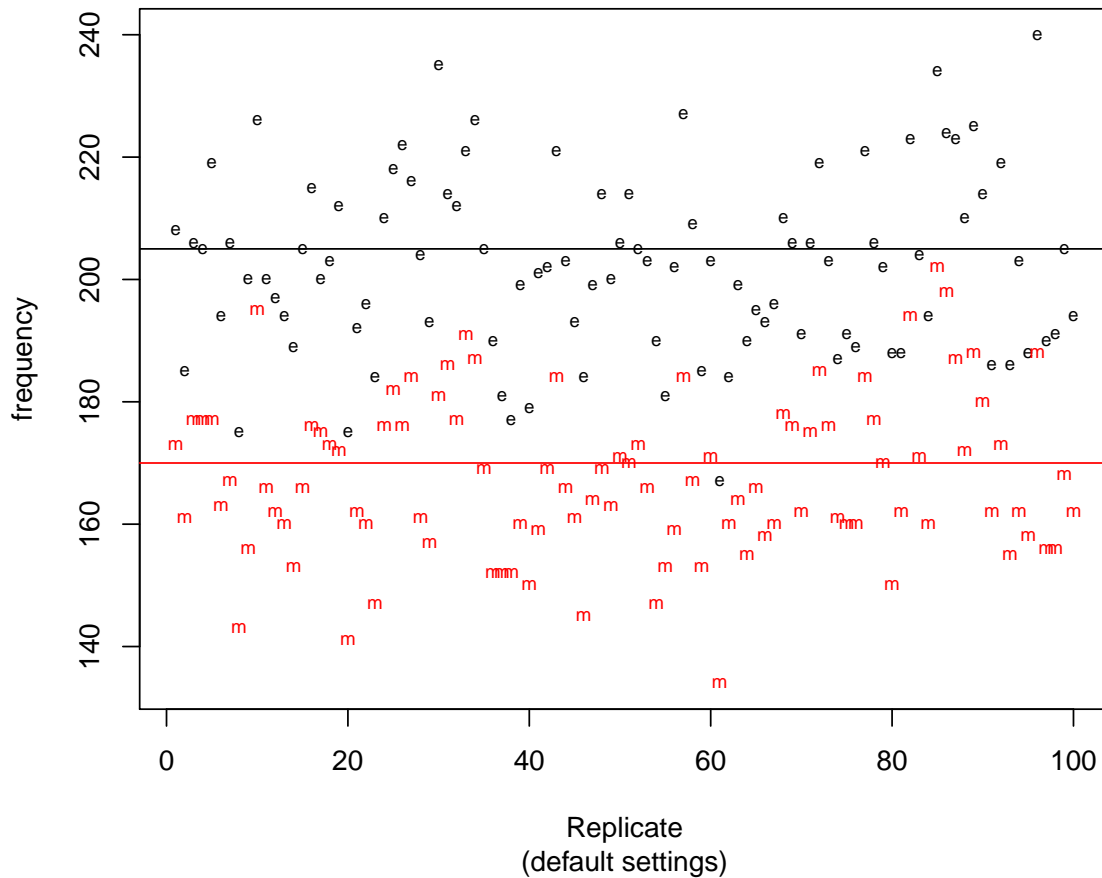
```
sim.stats <- attr(ego.sim100,"stats")
rbind(colMeans(sim.stats), ego.target.stats)

##          edges nodematch.sex
##          202.14      167.54
## ego.target.stats 205.00      170.00
```

These look pretty good – the means of the simulated target stats are within 1% of the observed. We can plot the 100 replicates to see check the variation for any problematic patterns:

```
matplot(1:nrow(sim.stats), sim.stats,
        pch=c("e","m","0","+"), cex=.65,
        main="100 simulations from ego.fit model", sub="(default settings)",
        xlab="Replicate", ylab="frequency")
abline(h=ego.target.stats, col=c(1:4))
```

### 100 simulations from ego.fit model



The lines mark the target statistic frequencies in the observed data. The points represent the frequencies in the simulated networks.

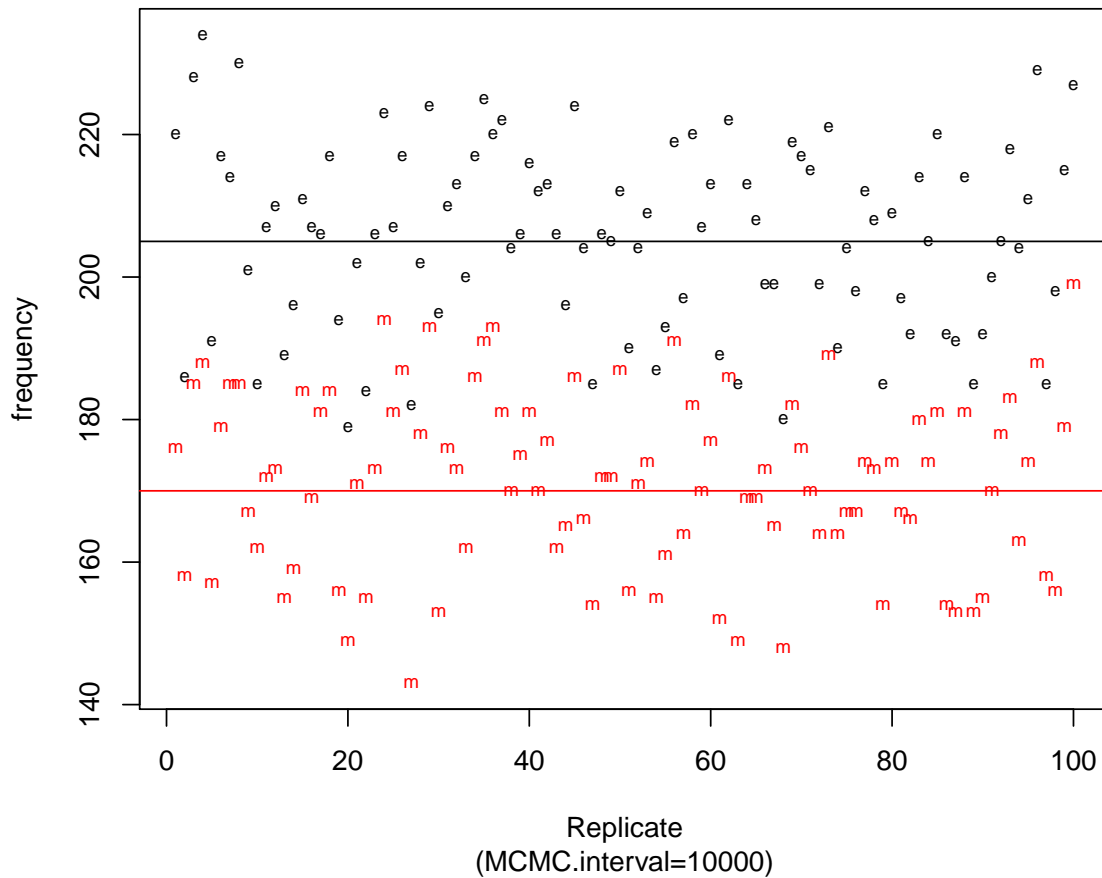
The simulated network statistics vary stochastically around the target values, not trending over time.

But, there is clear autocorrelation across the replicates, which suggests we might want to increase the MCMC interval to draw more independent realizations.

```
ego.sim100 <- simulate(ego.fit, nsim=100,
  control=control.simulate.ergm(MCMC.interval=10000))
sim.stats <- attr(ego.sim100,"stats")
matplot(1:nrow(sim.stats), sim.stats,
  pch=c("e","m"), cex=.65,
  main="100 simulations from ego.fit model", sub="(MCMC.interval=10000)",
  xlab="Replicate", ylab="frequency")
abline(h=ego.target.stats, col=c(1:2))
```



## 100 simulations from ego.fit model



With the larger interval, the autocorrelation is no longer detectable, and all of the statistics from the simulated networks vary in a symmetric band around their targets.

The variation (about  $\pm 10\%$ ) represents the range of target statistics that are consistent with the fitted coefficients.

If you wanted instead to constrain these statistics to equal a specified value, then you can use the “constraints=” argument during the `ergm` fit instead.

This is good for verifying that the simulation reproduces the target values we specified. So now let’s see whether the simulated complete networks also match statistics that were not set by the targets. We targeted edges and homophily. How well does this model reproduce the full degree distribution?

```
sim.fulldeg <- summary(ego.sim100 ~ degree(c(0:10)))
colnames(sim.fulldeg) <- paste("deg",0:10, sep='')
sim.fulldeg[1:5,]
```

	deg0	deg1	deg2	deg3	deg4	deg5	deg6	deg7	deg8	deg9	deg10
## [1,]	201	187	87	22	2	1	0	0	0	0	0
## [2,]	244	168	64	20	4	0	0	0	0	0	0
## [3,]	201	183	82	27	7	0	0	0	0	0	0
## [4,]	188	196	85	23	7	1	0	0	0	0	0

```
## [5,] 223 198 57 18 4 0 0 0 0 0 0
```

Recall that the degree range in our data was  $[0,3]$  by design, but we did not constrain the simulations to this range. If our model correctly captured the processes that led to the aggregate statistics we observe in our data, the simulated networks would show us what we missed. Here the simulated networks suggest that the fully observed network would have a wider range of degrees, which we might have observed, had we not truncated the data collection at 3 friends per respondent. About 1% of nodes have degree 4 or 5, and the max observed is 6.

But did our model did correctly capture the underlying processes? How well does the simulated degree distribution from this model match the frequencies we did observe? Aggregating the degrees of 3 or more in the simulations, we find:

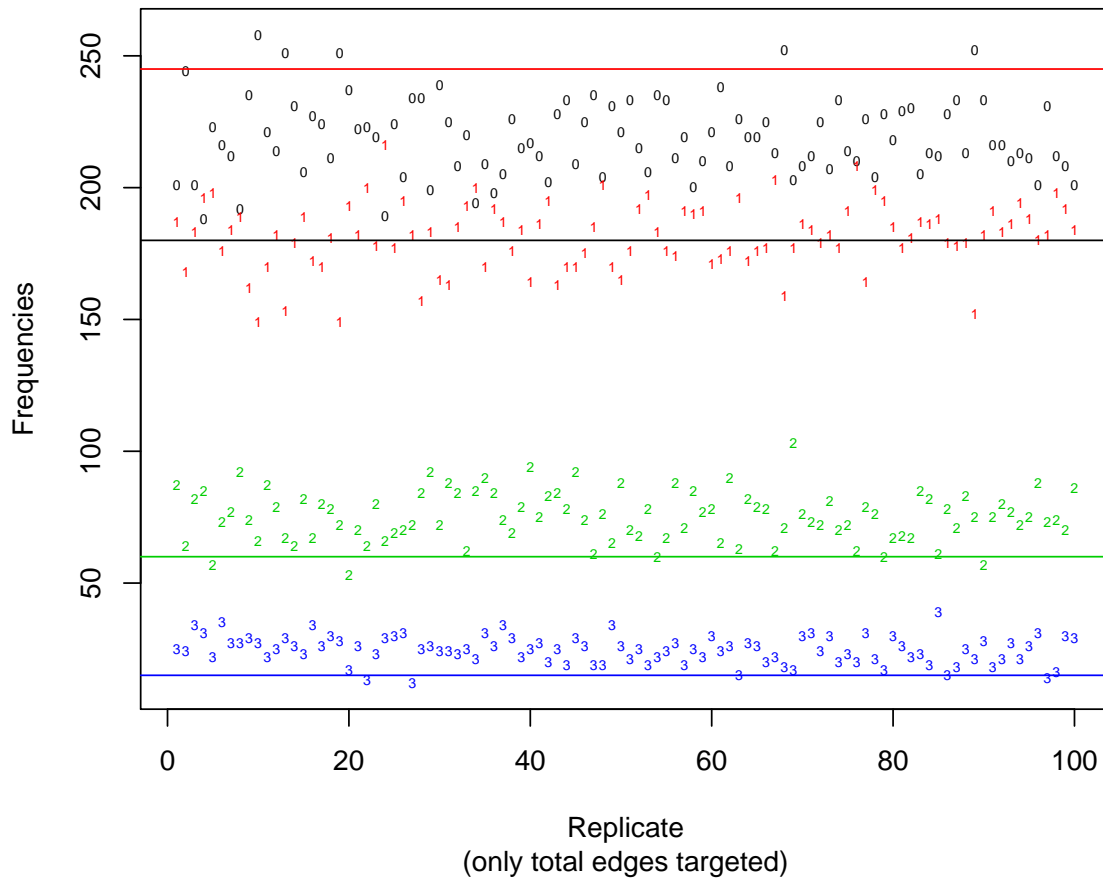
```
sim.deg <- cbind(sim.fulldeg[,1:3], apply(sim.fulldeg[,4:11],1,sum))
colnames(sim.deg) <- c(colnames(sim.fulldeg)[1:3], "degree3+")
rbind(colMeans(sim.deg), ego.deg)

##          deg0  deg1 deg2 degree3+
##          218.85 181.26 75.3    24.59
## ego.deg  180.00 245.00 60.0    15.00
```

As with the single simulation above, the discrepancies between the simulation averages and the observed statistics are quite large. We can see this more clearly by plotting the degree frequencies for the 100 replicate networks we simulated:

```
matplot(1:nrow(sim.deg), sim.deg, pch=as.character(0:3), cex=.5,
  main="Comparing ego.sims to non-targeted degree frequencies",
  sub = "(only total edges targeted)",
  xlab = "Replicate", ylab = "Frequencies")
abline(h=c(180, 245, 60, 15), col=c(1:4))
```

## Comparing ego.sims to non-targeted degree frequencies



The simulations are producing systematically more isolates than expected (the “0” points vs. the black line), and systematically more degree 1 nodes. In fact, the two degree frequencies are essentially reversed in the simulation.

The fraction of nodes with 2 or 3+ partners is systematically off but by a much smaller amount.

So our observed network has fewer isolates than expected in a network of this density, more degree 1 nodes than expected, and fewer degree 2 and 3+ nodes.

This suggests the model is mis-specified. Since the degree 0 vs. degree 1 is the worst fitting aspect, we will try using the number of isolates as a target statistic in the model.

```
ego.isolates <- ego.deg[1]
ego.target.stats <- c(ego.edges, ego.sexmatch, ego.isolates)
ego.fit <- ergm(ego.net ~ edges + nodematch('sex') + degree(0),
  target.stats = ego.target.stats)

## Starting maximum pseudolikelihood estimation (MPLE):
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Finished MPLE.
```

```

## Starting maximum pseudolikelihood estimation (MPLE):
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Finished MPLE.
## 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 30:
## Optimizing with step length 1.
## The log-likelihood improved by 0.02352.
## Convergence test p-value: 0.902985670435072. Not converged with 99% confidence; increasing
sample size.
## Iteration 2 of at most 30:
## Optimizing with step length 1.
## The log-likelihood improved by 0.02366.
## Convergence test p-value: 0.766204014974666. Not converged with 99% confidence; increasing
sample size.
## Iteration 3 of at most 30:
## Optimizing with step length 1.
## The log-likelihood improved by 0.01172.
## Convergence test p-value: 0.0525419435123568. Not converged with 99% confidence;
increasing sample size.
## Iteration 4 of at most 30:
## Optimizing with step length 1.
## The log-likelihood improved by 0.03147.
## Convergence test p-value: 0.601637150380275. Not converged with 99% confidence; increasing
sample size.
## Iteration 5 of at most 30:
## Optimizing with step length 1.
## The log-likelihood improved by 0.005059.
## Convergence test p-value: 3.40822134793939e-07. Converged with 99% confidence.
## 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 17 18 19 20 .
## This model was fit using MCMC. To examine model diagnostics and check for degeneracy,
use the mcmc.diagnostics() function.

summary(ego.fit)

##
## =====
## Summary of model fit
## =====
##
## Formula: nw ~ edges + nodematch("sex") + degree(0)
## <environment: 0x55c7698ca030>
##

```

```
## Iterations: 5 out of 30
##
## Monte Carlo MLE Results:
##           Estimate Std. Error MCMC % p-value
## edges      -8.4115    0.2452      0 <1e-04 ***
## nodematch.sex 1.5936    0.1811      0 <1e-04 ***
## degree0     -0.9635    0.1594      0 <1e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Null Deviance: 172940 on 124750 degrees of freedom
## Residual Deviance: 2903 on 124747 degrees of freedom
##
## AIC: 2909 BIC: 2938 (Smaller is better.)
```

Simulating from this model, the target statistics are again well matched:

```
ego.sim100 <- simulate(ego.fit, nsim=100,
  control=control.simulate.ergm(MCMC.interval=10000))
sim.stats <- attr(ego.sim100,"stats")
rbind(colMeans(sim.stats), ego.target.stats)

##           edges nodematch.sex degree0
##           205.58      169.98 179.07
## ego.target.stats 205.00      170.00 180.00
```

And the full degree frequencies look much better:

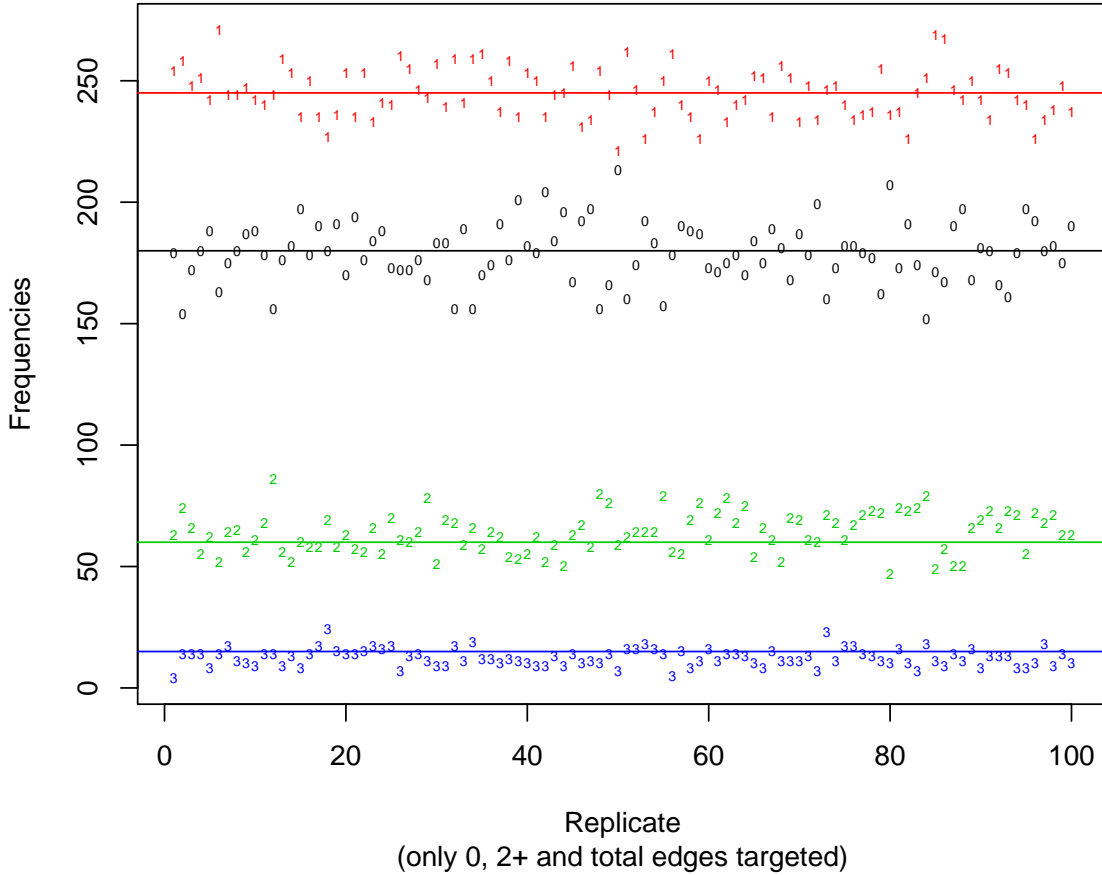
```
sim.fulldeg <- summary(ego.sim100 ~ degree(c(0:10)))
sim.deg <- cbind(sim.fulldeg[,1:3], apply(sim.fulldeg[,4:11],1,sum))
colnames(sim.deg) <- c(colnames(sim.fulldeg)[1:3], "degree3+")
rbind(colMeans(sim.deg), ego.deg)

##           degree0 degree1 degree2 degree3+
##           179.07   244.7   63.79   12.44
## ego.deg  180.00   245.0   60.00   15.00
```

and finally, plotting the full degree frequencies

```
matplot(1:nrow(sim.deg), sim.deg, pch=as.character(0:3), cex=.5,
  main="Comparing ego.sims to non-targeted degree frequencies",
  sub = "(only 0, 2+ and total edges targeted)",
  xlab = "Replicate", ylab = "Frequencies")
abline(h=c(180, 245, 60, 15), col=c(1:4))
```

## Comparing ego.sims to non-targeted degree frequencies



The degree frequencies in these simulated networks are now well centered on the observed frequencies. So adding the one additional parameter to capture the lower than expected number of isolates did a good job of capturing how our observed network deviates from a random network with this density.

The fraction of nodes with 3+ partners produced by our new model might still be a bit low.

**Moral:** We can use ERGMs to estimate network models using target statistics from egocentrically sampled data. The fact that the target statistics are reproduced by this model does not guarantee that additional features of the network would also be reproduced. But starting with simple models can help to identify whether and how the aggregate statistics we observe from an egocentric sample deviate from those we would expect from the model. If we fit all of the observed statistics without a saturated model, we cannot reject the hypothesis that this model produced the network we sampled from.

We can also use this approach to explore network statistics that are not visible at all from the egocentric data, e.g., the geodesic distribution, betweenness, etc., but it must always be remembered that the distributions we will produce are based on our model. They faithfully reproduce the model, but that does not mean that the model faithfully represents the population.

In the STERGM workshop, we show how complete dynamic networks also can be simulated over time on the basis of egocentric data like these, with the minimal addition of a single estimate of partnership duration. For a movie of an example dynamic simulation used to explore the impact of network structure on HIV transmission, see [statnet.org/movies](http://statnet.org/movies).

## 8 Additional functionality in the statnet family of packages

### 8.1 In the `ergm` and `network` packages:

- ERGMs for valued ties – see the paper by Pavel Krivitsky (2012)
- Analysis of bipartite networks – `network` recognizes this as an attribute of the network and `ergm` provides specific model terms for such networks that begin with `b1` or `b2` (try: `search.ergmTerms(categories=c('bipartite'))`).

### 8.2 In other packages from the statnet suite:

These are in stand-alone packages that can be downloaded either from CRAN, or from the `statnet` website. Many have online training materials from our workshops. For more detailed information, please visit the `statnet` webpage (<http://statnet.org>).

#### 8.2.1 Static (cross-sectional) network analysis packages

- `sna` – Traditional SNA methods and summaries.
- `latentnet` – Latent space and latent cluster analysis.
- `netperm` – Network permutation models.
- `degreeenet` – MLE estimation for degree distributions (negative binomial, Poisson, scale-free, etc.)
- `networksis` – Simulation of bipartite networks with given degree distributions.

#### 8.2.2 Dynamic (longitudinal) network analysis packages:

- `tergm` – Temporal ERGMs (TERGMs): discrete-time dynamic network models for longitudinal network panel data, and other temporal extensions.
- `relevent` – Relational event models: continuous-time dynamic network models for longitudinal network data.
- `networkDynamic` – Dynamic network data storage and manipulation.
- `ndtv` – Network movie maker.

### 8.3 R packages that build on statnet

There is a growing number of R packages written by other folks that build on or extend the functionality of the `statnet` suite. You can get a current list of those packages by looking at the reverse depends/suggests on CRAN. A partial list includes:

- `EpiModel` package – Mathematical Modeling of Infectious Disease, includes functions for deterministic compartmental modeling, stochastic individual contact modeling, and stochastic network modeling
- `RDS` package – Estimation with data collected using Respondent-Driven Sampling.
- `Bergm` package – Bayesian ERGM estimation

- **hergm** package – Hierarchical Exponential-Family Random Graph Models with Local Dependence (for latent groups).
- **lvm4net** package – Latent variable models.
- **VBLPCM** package – Variational Bayes Latent Position Cluster Models.
- **xergm** package – Temporal exponential random graph models (TERGM) by bootstrapped pseudolikelihood, MCMC MLE and (temporal) network autocorrelation models.

## 8.4 Additional functionality in development:

- **ergm.ego** package – ERGM estimation and inference from egocentrically sampled data (expected May 2015)
- **tsna** package – Temporally extended (vertex and edge) SNA methods for dynamic longitudinal network data (expected May 2015)
- **MLergm** package – ERGM estimation and inference for multi-level data (for observed groups) (expected 2016)

## 9 Statnet Commons: The core development team

Mark S. Handcock [jhandcock@stat.ucla.edu](mailto:jhandcock@stat.ucla.edu)  
 David R. Hunter [jdhunter@stat.psu.edu](mailto:jdhunter@stat.psu.edu)  
 Carter T. Butts [jbuttsc@uci.edu](mailto:jbuttsc@uci.edu)  
 Steven M. Goodreau [jgoodreau@u.washington.edu](mailto:jgoodreau@u.washington.edu)  
 Skye Bender-deMoll [jskyebend@skyeome.net](mailto:jskyebend@skyeome.net)  
 Martina Morris [jmorris@u.washington.edu](mailto:jmorris@u.washington.edu)  
 Pavel N. Krivitsky [jpavel@uow.edu.au](mailto:jpavel@uow.edu.au)

## Appendix A: Clarifying the terms – ergm and network

You will see the terms **ergm** and **network** used in multiple contexts throughout the documentation. This is common in R, but often confusing to newcomers. To clarify:

### **ergm**

- **ERGM**: the acronym for an Exponential Random Graph Model; a statistical model for relational data that takes a generalized exponential family form.
- **ergm package**: one of the packages within the **statnet** suite
- **ergm function**: a function within the **ergm** package; fits an ERGM to a network object, creating an **ergm** object in the process.
- **ergm object**: a class of objects produced by a call to the **ergm** function, representing the results of an ERGM fit to a network.

### **network**

- **network**: a set of actors and the relations among them. Used interchangeably with the term **graph**.



- **network package**: one of the packages within the **statnet** suite; used to create, store, modify and plot the information found in network objects.
- **network object**: a class of object in R used to represent a network.

## References

- Goodreau, S., J. Kitts and M. Morris (2009). Birds of a Feather, or Friend of a Friend? Using Statistical Network Analysis to Investigate Adolescent Social Networks. *Demography* 46(1):103-125.
- Handcock, M. S., D. R. Hunter, C. T. Butts, S. M. Goodreau and M. Morris (2008). statnet: Software Tools for the Representation, Visualization, Analysis and Simulation of Network Data. *Journal of Statistical Software* 42(01).
- Hunter DR, Handcock MS, Butts CT, Goodreau SM, Morris M (2008b). ergm: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks. *Journal of Statistical Software*, 24(3). <http://www.jstatsoft.org/v24/i03/>.
- Krivitsky, P.N.(2009). PhD thesis. *University of Washington, Seattle, WA*
- Krivitsky, P. N., M. S. Handcock and M. Morris (2011). Network Size and Composition Effects in Exponential-Family Random Graph Models. *Statistical Methodology* 8:319-339
- Krivitsky PN (2012). Exponential-family random graph models for valued networks. *Electronic Journal of Statistics* 6:1100-1128