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

Version 3.10.1

The Statnet Development Team

May 14, 2019

Contents

1 Getting Started **1**

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

3 Model terms available for *ergm* estimation and simulation **10**
3.1 Terms provided with ergm 11
3.2 Coding new terms 11

4 Network simulation: the *simulate* command and *network.list* objects **11**

5 Examining the quality of model fit – *GOF* **13**

6 Diagnostics: troubleshooting and checking for model degeneracy **17**

7 Working with egocentrically sampled network data **26**

8 Additional functionality in the statnet family of packages **27**
8.1 In the `ergm` and `network` packages: 27
8.2 In other packages from the statnet suite: 27
8.2.1 Static (cross-sectional) network analysis packages 27
8.2.2 Dynamic (longitudinal) network analysis packages: 28
8.3 R packages that build on statnet 28
8.4 Additional functionality in development: 28

9 Statnet Commons: The core development team **28**

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

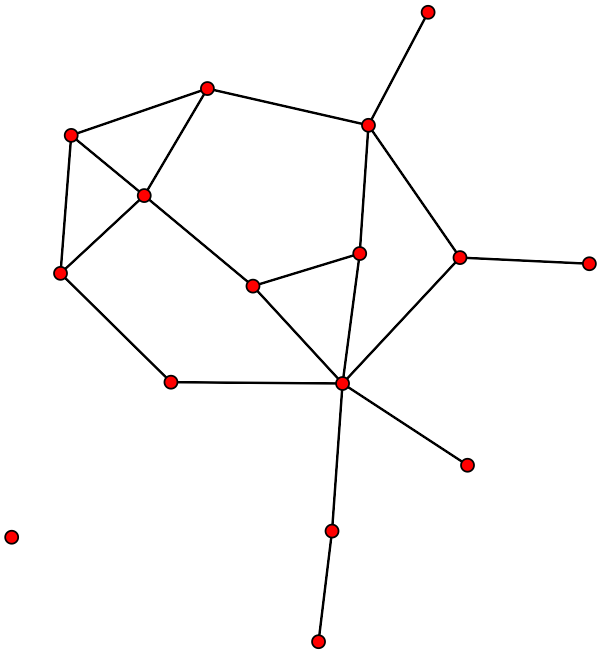
```
library(ergm)
library(sna)
```

```
## 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; θ is the vector of coefficients; $k(\theta)$ is a normalizing constant.

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

Y_{ij} is an actor pair in Y; Y^c is the rest of the network; $\Delta \left(g(y)\right)_{ij}$ 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 Erd?s-R?nyi 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.
## Stopping at the initial estimate.
## 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 20
##
## Monte Carlo MLE Results:
##       Estimate Std. Error MCMC % z value Pr(>|z|)
## edges  -1.6094     0.2449      0  -6.571   <1e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 166.4  on 120  degrees of freedom
##  Residual Deviance: 108.1  on 119  degrees of freedom
##
## AIC: 110.1    BIC: 112.9    (Smaller is better.)
```

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 20:
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## Optimizing with step length 1.
## The log-likelihood improved by 0.002363.
## Step length converged once.  Increasing MCMC sample size.
## Iteration 2 of at most 20:
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## Optimizing with step length 1.
## The log-likelihood improved by 0.001553.
## Step length converged twice.  Stopping.
## Finished MCMLE.
## Evaluating log-likelihood at the estimate.  Using 20 bridges:  1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 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:  2 out of 20
##
## Monte Carlo MLE Results:
##          Estimate Std. Error MCMC % z value Pr(>|z|)
## edges     -1.6793     0.3496      0  -4.804   <1e-04 ***
## triangle   0.1580     0.5697      0   0.277    0.782
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 166.4  on 120  degrees of freedom
##  Residual Deviance: 108.1  on 118  degrees of freedom
##
## AIC: 112.1    BIC: 117.7    (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:

$$-1.679 \times \text{change in the number of ties} + 0.158 \times \text{change in number of triangles}$$

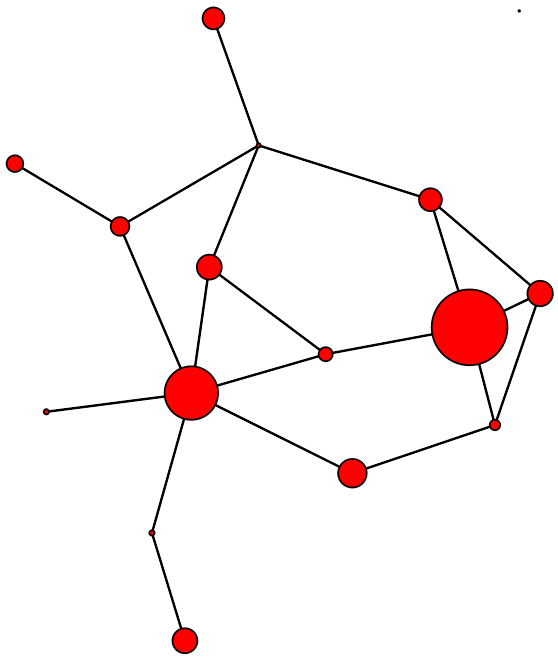
- if the tie will not add any triangles to the network, its log-odds is: -1.679.
- if it will add one triangle to the network, its log-odds is: $-1.679 + 0.158 = -1.521$
- if it will add two triangles to the network, its log-odds is: $-1.679 + 0.158 \times 2 = -1.363$
- the corresponding probabilities are 0.157, 0.179, and 0.204.

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

```
flomodel.02$coef

##      edges   triangle
## -1.6792968  0.1579912

flomodel.02$formula

## flomarriage ~ edges + triangle

flomodel.02$mle.lik
```

```
## 'log Lik.' -54.03959 (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.
## Stopping at the initial estimate.
## Evaluating log-likelihood at the estimate.

summary(flomodel.03)

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

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)
sampmodel.01 <- ergm(samplk3~edges+mutual)
summary(sampmodel.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 size 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`)

3.2 Coding new terms

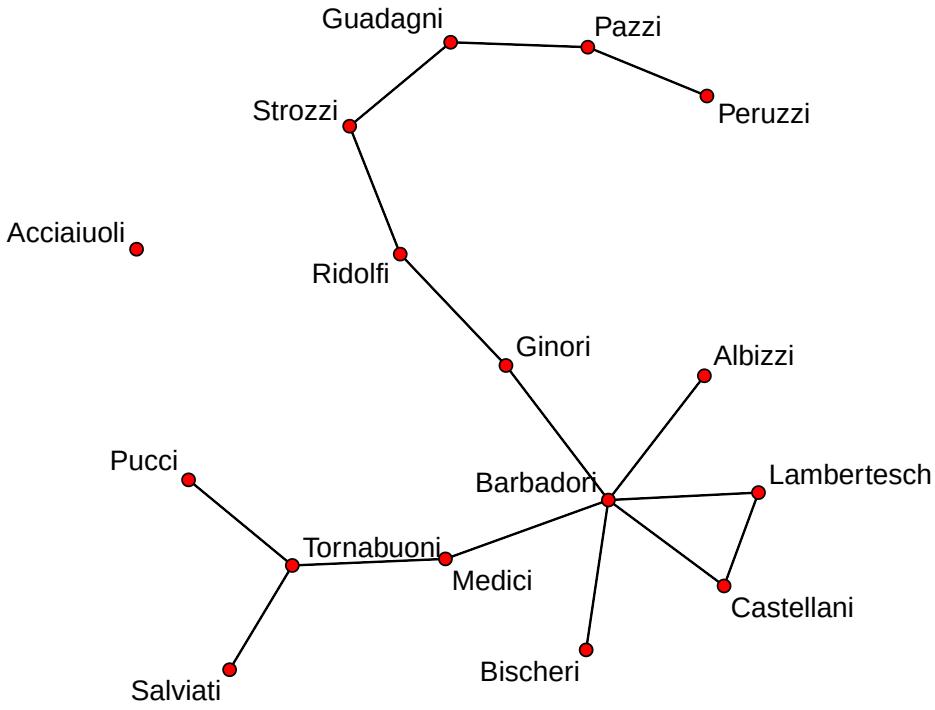
The package (`ergm.userterms`) and tutorial are aimed at making it much easier than before to write one's own terms. This package is available on CRAN, and installing it will also download the tutorial (ergmuserterms.pdf). We teach a workshop at the Sunbelt meetings. Note that writing up new `ergm` terms 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,]    15           1468
##  [2,]    26           2231
```

```
##  [3,]    20           1975
##  [4,]    24           2550
##  [5,]    25           2684
##  [6,]    21           2460
##  [7,]    16           1527
##  [8,]    14           1233
##  [9,]    19           2204
## [10,]    20           1817
## attr(,"monitored")
## [1] FALSE FALSE
## 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= 15
##     missing edges= 0
##     non-missing edges= 15
##
##  Vertex attribute names:
##     priorates totalties vertex.names wealth
##
## No edge attributes

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



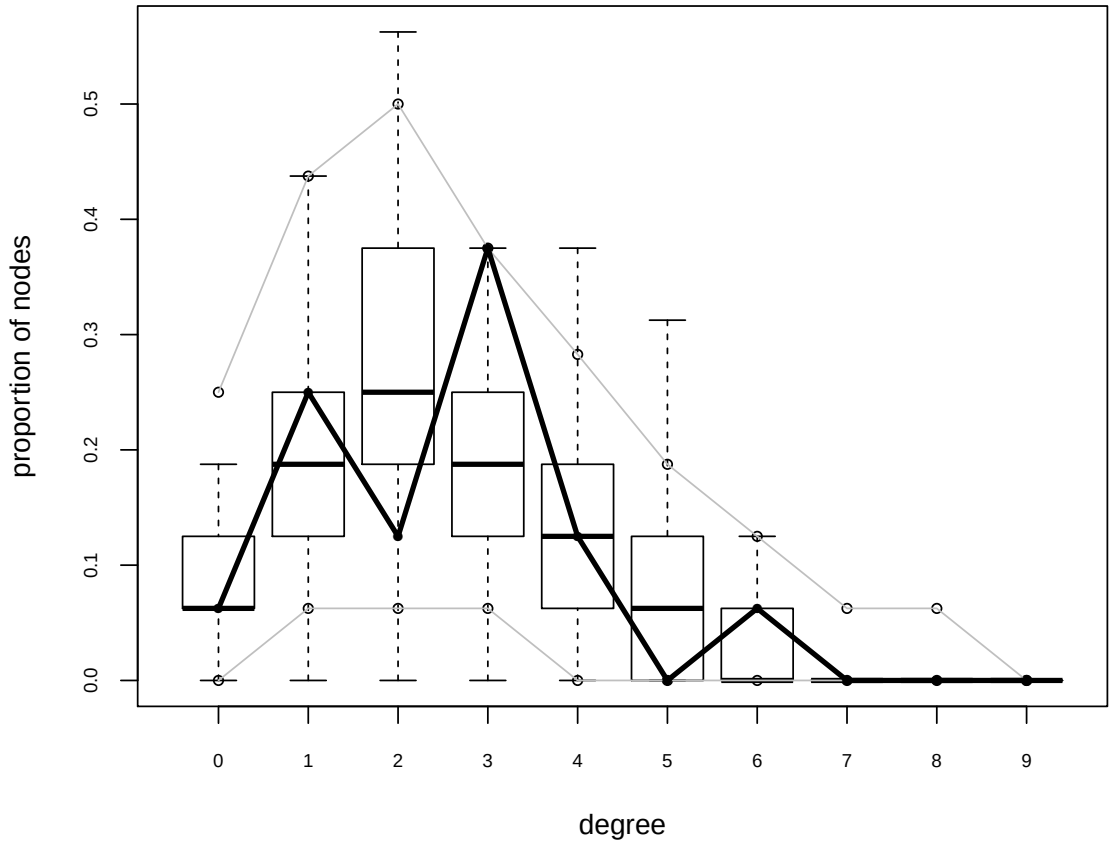
Voila. Of course, yours will look somewhat different.

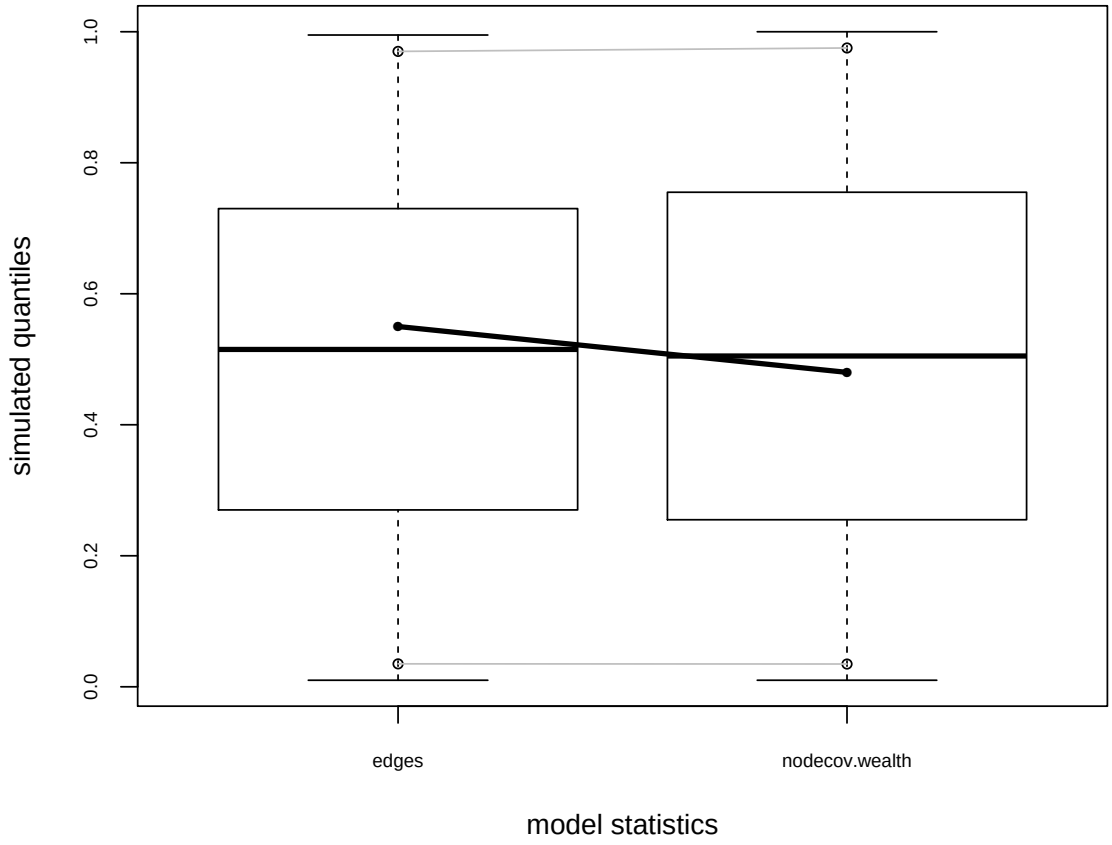
5 Examining the quality of model fit – *GOF*

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.

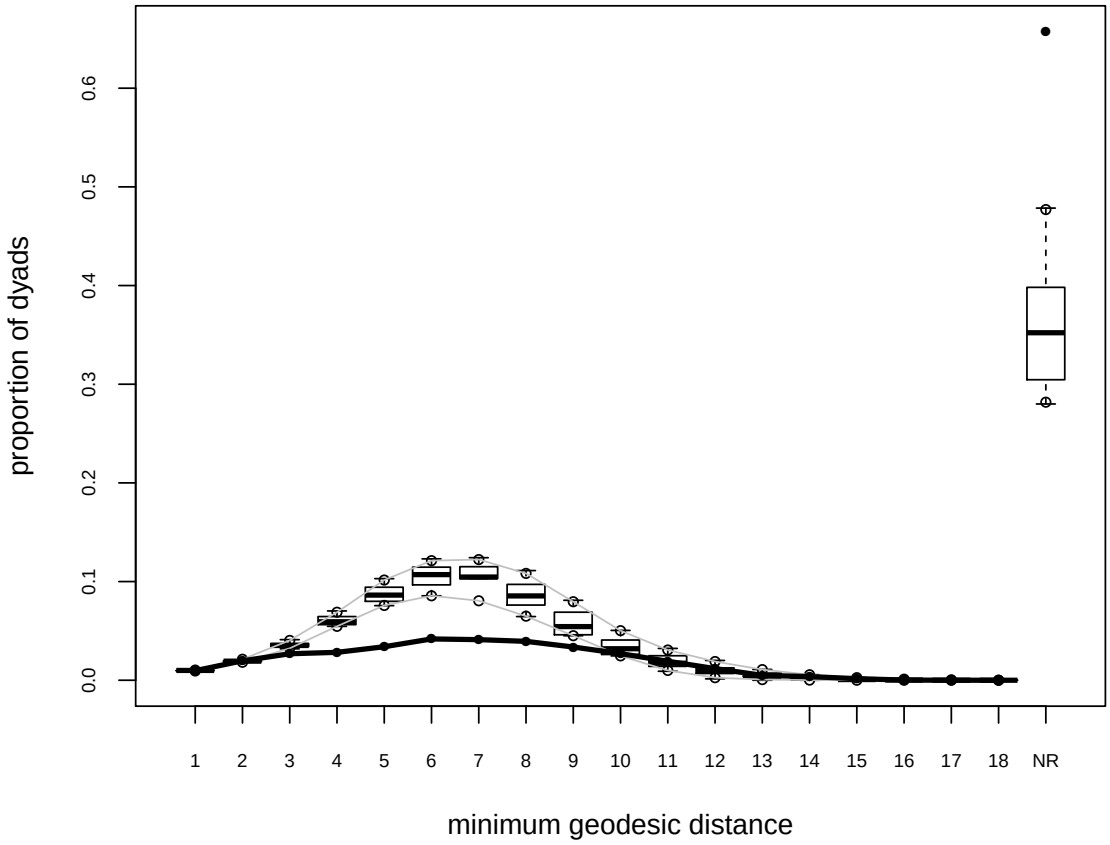
```
flomodel.03.gof <- gof(flomodel.03~degree)

flomodel.03.gof

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

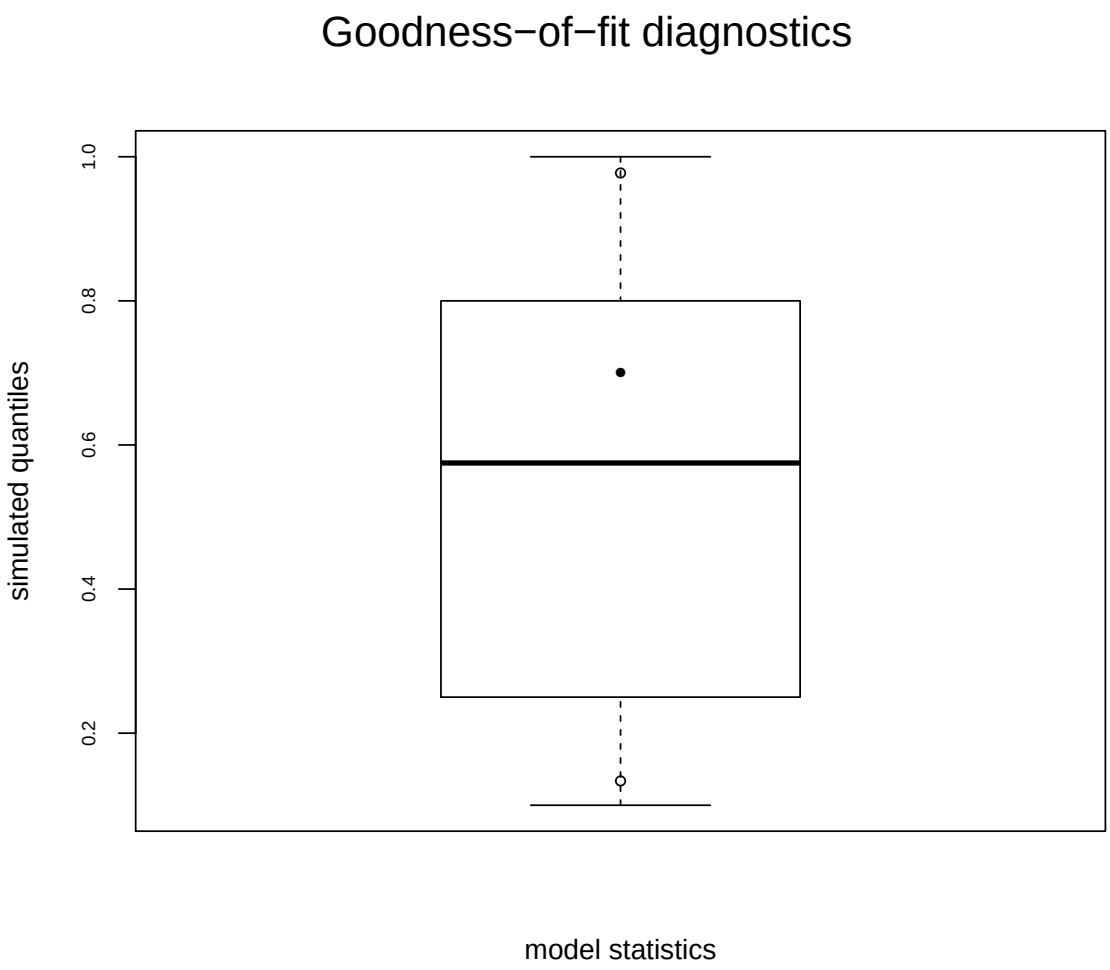
```
## 0   1   0 1.38   4       1.00
## 1   4   0 3.26   9       0.82
## 2   2   0 4.36   9       0.34
## 3   6   0 3.25   9       0.22
## 4   2   0 1.88   6       1.00
## 5   0   0 1.09   5       0.64
## 6   1   0 0.49   2       0.82
## 7   0   0 0.22   2       1.00
## 8   0   0 0.05   1       1.00
## 9   0   0 0.02   1       1.00
##
## Goodness-of-fit for model statistics
##
##                 obs  min    mean  max MC p-value
## edges            20   11   19.88   28       1.00
## nodecov.wealth 2168 1023 2125.99 3045       0.96

plot(flomodel.03.gof)
```






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

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

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


proportion of dyads
0.0
0.1
0.2
0.3
0.4
0.5
0.6
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
NR
minimum geodesic distance



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`)

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:

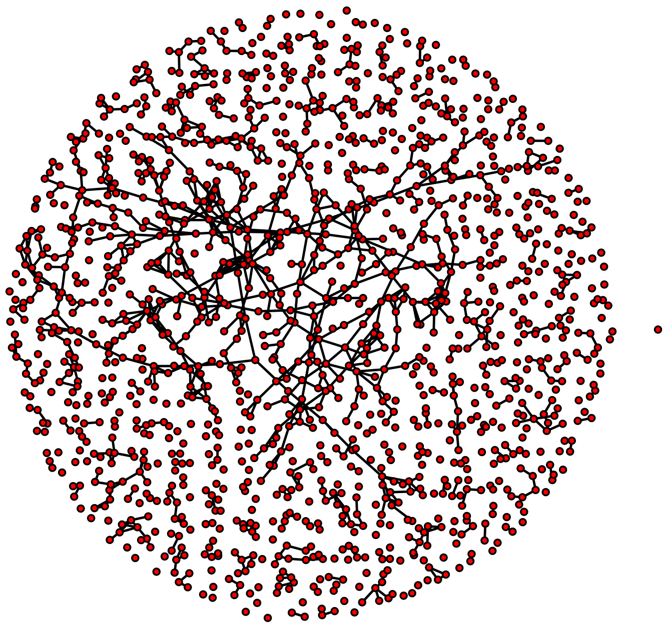
```
mcmc.diagnostics(fit, center=F)
```

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))

## Starting maximum pseudolikelihood estimation (MPLE):
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Finished MPLE.
## Starting Monte Carlo maximum likelihood estimation (MCMLE):
## Iteration 1 of at most 20:
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
```

```
## NOTE: Messages 'Error in mcexit(OL)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(OL)' may appear; please disregard them.
## Optimizing with step length 0.456687805180042.
## The log-likelihood improved by 3.541.
## Iteration 2 of at most 20:
## NOTE: Messages 'Error in mcexit(OL)' may appear; please disregard them.
## Optimizing with step length 1.
## The log-likelihood improved by 1.869.
## Step length converged once.  Increasing MCMC sample size.
## Iteration 3 of at most 20:
## Error in ergm.MCMLE(init, nw, model, initialfit = (initialfit <- NULL), :  Number of
edges in a simulated network exceeds that in the observed by a factor of more than 20.
This is a strong indicator of model degeneracy or a very poor starting parameter configuration.
If you are reasonably certain that neither of these is the case, increase the MCMLE.density.guard
control.ergm() parameter.
```

Very interesting. 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))
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),
 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')+
```

```r
       nodematch('Sex'),
 control = control.ergm(seed=1))
mcmc.diagnostics(fit)
```

One more try...

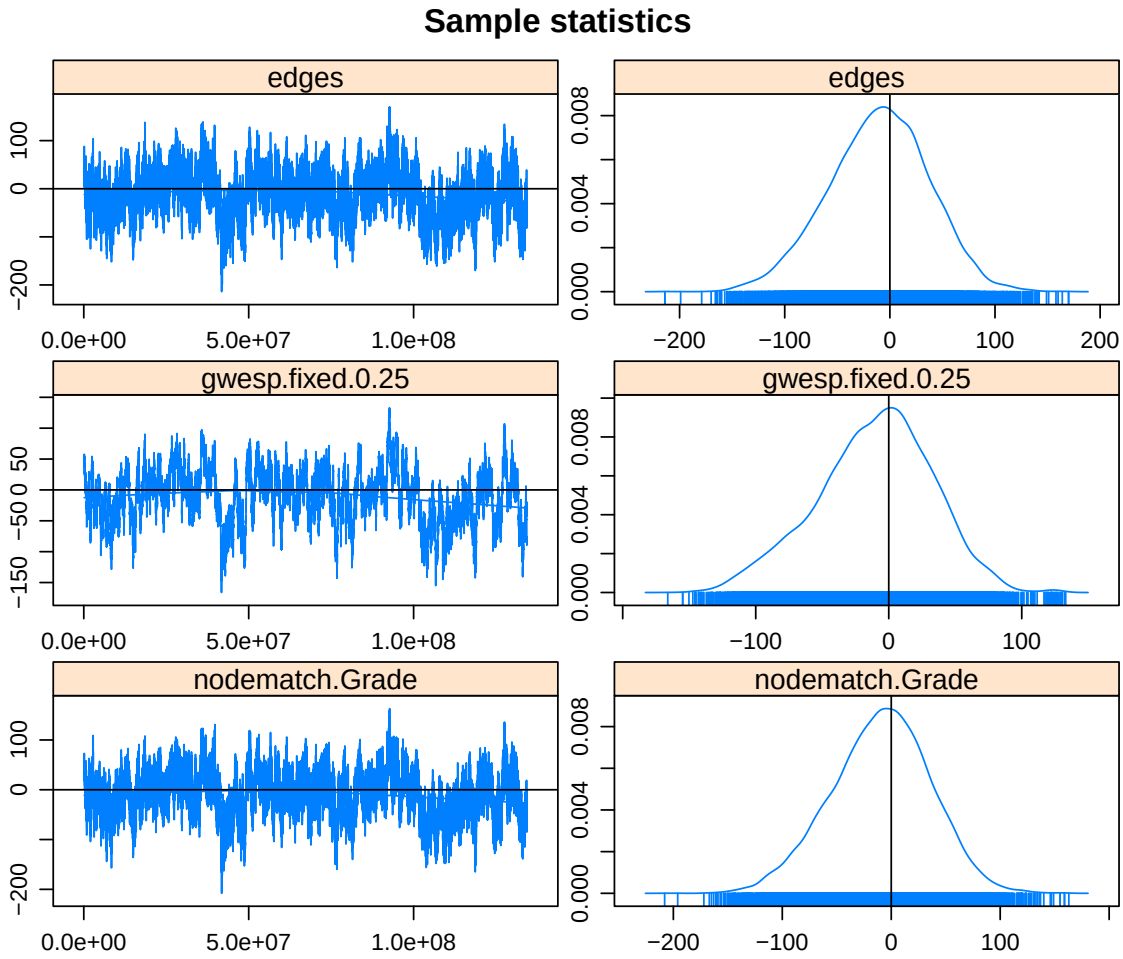
```r
fit <- ergm(magnolia~edges+gwesp(0.25,fixed=T)+nodematch('Grade')+nodematch('Race')+
  nodematch('Sex'),
 control = control.ergm(seed=1,MCMC.samplesize=4096,MCMC.interval=8192),
 verbose=T)

## Evaluating network in model.
## Initializing Metropolis-Hastings proposal(s):  ergm:MH_TNT
## Initializing model.
## Using initial method 'MPLE'.
## Fitting initial model.
## Starting maximum pseudolikelihood estimation (MPLE):
## Evaluating the predictor and response matrix.
## 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 20 with parameter:
##             edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##        -9.8619687        1.6946112        2.8534613        0.9886313
##    nodematch.Sex
##        0.8245285
## Starting unconstrained MCMC...
## Back from unconstrained MCMC.
## Average estimating function values:
##             edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##         132.76782         73.53185        131.48535        128.23169
##    nodematch.Sex
##         115.33716
## Starting MCMLE Optimization...
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## NOTE: Messages 'Error in mcexit(0L)' may appear; please disregard them.
## Optimizing with step length 0.969866328489393.
## Using lognormal metric (see control.ergm function).
## Using log-normal approx (no optim)
## The log-likelihood improved by 5.555.
```

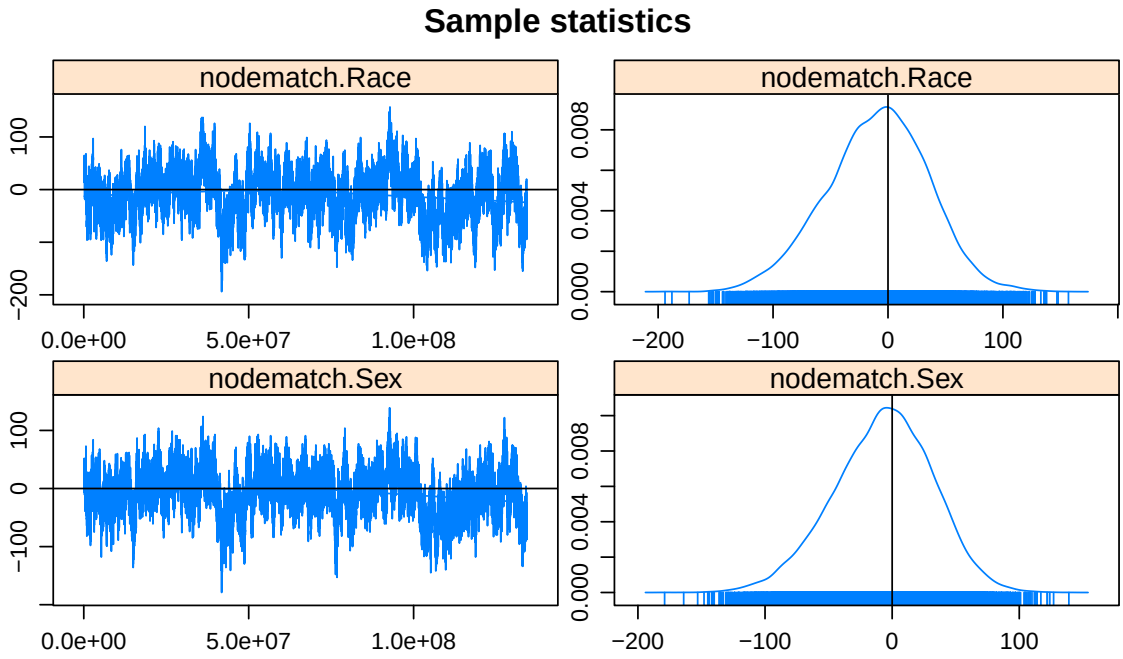
```
##
## Iteration 2 of at most 20 with parameter:
##            edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      -9.7936042        1.7967374        2.7511699        0.9226989
##    nodematch.Sex
##        0.7713285
## Starting unconstrained MCMC...
## Back from unconstrained MCMC.
## Average estimating function values:
##            edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      -0.01391602      -2.22583659       0.61816406      2.39648438
##    nodematch.Sex
##      1.03515625
## Starting MCMLE Optimization...
## NOTE: Messages 'Error in mcexit(OL)' may appear; please disregard them.
## Optimizing with step length 1.
## Using lognormal metric (see control.ergm function).
## Using log-normal approx (no optim)
## The log-likelihood improved by 0.03205.
## Step length converged once.  Increasing MCMC sample size.
##
## Iteration 3 of at most 20 with parameter:
##            edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##      -9.7756172        1.8051362        2.7422501        0.9085738
##    nodematch.Sex
##        0.7657380
## Starting unconstrained MCMC...
## Back from unconstrained MCMC.
## Average estimating function values:
##            edges gwesp.fixed.0.25  nodematch.Grade  nodematch.Race
##        10.667053        10.684350        9.603271        9.508240
##    nodematch.Sex
##        7.205261
## Starting MCMLE Optimization...
## NOTE: Messages 'Error in mcexit(OL)' may appear; please disregard them.
## 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 0.03392.
## Step length converged twice.  Stopping.
## Finished MCMLE.
## Evaluating log-likelihood at the estimate.  Using 20 bridges:  1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 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 = 131072:134340608
## Thinning interval = 8192
## Number of chains = 1
## Sample size per chain = 16384
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                      Mean    SD Naive SE Time-series SE
## edges             -10.667 47.81   0.3735          4.174
## gwesp.fixed.0.25 -10.684 43.54   0.3402          4.410
## nodematch.Grade   -9.603 45.59   0.3562          4.078
## nodematch.Race    -9.508 44.36   0.3466          4.333
## nodematch.Sex     -7.205 38.97   0.3044          3.562
##
## 2. Quantiles for each variable:
##
##                     2.5%    25%    50%   75% 97.5%
## edges             -108.0 -42.00 -9.000 22.00 79.00
## gwesp.fixed.0.25 -102.3 -38.57 -7.678 19.52 68.63
## nodematch.Grade  -103.0 -39.00 -8.000 22.00 75.42
## nodematch.Race   -100.4 -38.00 -8.000 21.00 74.00
## nodematch.Sex     -88.0 -33.00 -5.000 20.00 65.00
##
##
## Sample statistics cross-correlations:
##                      edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
## edges            1.0000000        0.8556558       0.9631021      0.9491737
## gwesp.fixed.0.25 0.8556558        1.0000000       0.8762310      0.8516263
## nodematch.Grade  0.9631021        0.8762310       1.0000000      0.9225765
## nodematch.Race   0.9491737        0.8516263       0.9225765      1.0000000
## nodematch.Sex    0.9109089        0.8032580       0.8827915      0.8668522
##                  nodematch.Sex
## edges                0.9109089
## gwesp.fixed.0.25     0.8032580
## nodematch.Grade      0.8827915
## nodematch.Race       0.8668522
## 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 8192  0.7863081        0.9830872       0.8366096      0.8260462
## Lag 16384 0.7411780        0.9677757       0.7908314      0.7794904
## Lag 24576 0.7192838        0.9533507       0.7687282      0.7558382
## Lag 32768 0.7035303        0.9398111       0.7516197      0.7378241
## Lag 40960 0.6887798        0.9268211       0.7388474      0.7238515
##           nodematch.Sex
## Lag 0         1.0000000
## Lag 8192      0.7968229
```

```
## Lag 16384     0.7475870
## Lag 24576     0.7223195
## Lag 32768     0.7031249
## Lag 40960     0.6904162
##
## 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
##          -0.5346          -0.4654          -0.5177          -0.6900
##    nodematch.Sex
##           0.1811
##
## Individual P-values (lower = worse):
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##        0.5929236        0.6416232        0.6046701        0.4901875
##    nodematch.Sex
##        0.8563071
## Joint P-value (lower = worse):  0.4058461 .
```


Sample statistics
edges
edges
gwesp.fixed.0.25
gwesp.fixed.0.25
nodematch.Grade
nodematch.Grade
0.0e+00
5.0e+07
1.0e+08
−200
−100
0
100
200
0.000
0.004
0.008




```
##
## 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

One of the most powerful features of ERGMs is that they can be used to estimate models from from egocentrically sampled data, and the fitted models can then be used to simulate complete networks (of any size) that will have the properties of the original network that are observed and represented in the model.

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.

The egocentric estimation/simulation framework extends to temporal ERGMs ("TERGMs") as well, with the minimal addition of an estimate of partnership duration. This makes it possible to simulate complete dynamic networks from a single cross-sectional egocentrically sampled network. For an example of what you can do with this, check out the network movie we developed to explore the impact of dynamic network structure on HIV transmission, see https://statnet.org/movies .

While the ergm package has had this capability for many years (and old ERGM worshops had a section on this), there is now a specific package that makes this much easier: 'ergm.ego'. The new package includes accurate statistical inference (so you can get standard errors for model coefficients), and many utilities that simplify the task of reading in the data, conducting exploratory analyses, and specifying model options.

We now have a separate workshop/tutorial for 'ergm.ego', so we no longer cover this material in the current ERGM workshop. As always, this workshop material can be found online at the statnet wiki.

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 (https://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.
- `degreenet` – 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 stochatic 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 ¡handcock@stat.ucla.edu¿
David R. Hunter ¡dhunter@stat.psu.edu¿
Carter T. Butts ¡buttsc@uci.edu¿
Steven M. Goodreau ¡goodreau@u.washington.edu¿
Skye Bender-deMoll ¡skyebend@skyeome.net¿
Martina Morris ¡morrism@u.washington.edu¿
Pavel N. Krivitsky ¡pavel@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). `https://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