Exponential Random Graph Models (ERGMs) using statnet

Sunbelt 2015 - Brighton, UK

Contents

Last updated 06-15-2015

This tutorial is a joint product of the Statnet Development Team:

Martina Morris (University of Washington)
Mark S. Handcock (University of California, Los Angeles)
Carter T. Butts (University of California, Irvine)
David R. Hunter (Penn State University)
Steven M. Goodreau (University of Washington)
Skye Bender de-Moll (Oakland)
Pavel N. Krivitsky (University of Wollongong)

For general questions and comments, please refer to statnet users group and mailing list
http://statnet.csde.washington.edu/statnet_users_group.shtml

1. Getting Started

Open an R session, and set your working directory to the location where you would like to save this work.

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('ergm') # will install the network package
install.packages('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 use 1 more package (latticeExtra), which is recommended (but not required) by ergm:

```
install.packages('latticeExtra')
```

Make sure the packages are attached:

```
library(statnet)
```

or

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

Check package version

```
# latest versions:  ergm 3.7.1 and network 1.13.0 (as of 7/24/2017)
sessionInfo()
```

Set seed for simulations – this is not necessary, but it ensures that we all get the same results (if we execute the same commands in the same order).

```
set.seed(0)
```

2. Statistical network modeling; the *summary* and *ergm* commands, and supporting functions

Exponential-family random graph models (ERGMs) represent a general class of models based in exponential-family theory for specifying the probability distribution for a set of random graphs or networks. Within this framework, one can—among other tasks—obtain maximum-likehood estimates for the parameters of a specified model for a given data set; test individual models for goodness-of-fit, perform various types of model comparison; and simulate additional networks with the underlying probability distribution implied by that model.

The general form for an ERGM can be written as:

$$P(Y = y) = \frac{\exp(\theta' g(y))}{k(\theta)}$$

where Y is the random variable for the state of the network (with realization y), $g(y)$ is a vector of model statistics for network y, θ is the vector of coefficients for those statistics, and $k(\theta)$ represents the quantity in the numerator summed over all possible networks (typically constrained to be all networks with the same node set as y).

This can be re-expressed in terms of the conditional log-odds of a single tie between two actors:

$$\text{logit}\,(Y_{ij} = 1 | y_{ij}^c) = \theta' \delta(y_{ij})$$

where Y_{ij} is the random variable for the state of the actor pair i, j (with realization y_{ij}), and y_{ij}^c signifies the complement of y_{ij}, i.e. all dyads in the network other than y_{ij}. The vector $\delta(y_{ij})$ contains the "change statistic" for each model term. The change statistic records how $g(y)$ term changes if the y_{ij} tie is toggled on or off. So:

$$\delta(y_{ij}) = g(y_{ij}^+) - g(y_{ij}^-)$$

where y_{ij}^+ is defined as y_{ij}^c along with y_{ij} set to 1, and y_{ij}^- is defined as y_{ij}^c along with y_{ij} set to 0. That is, $\delta(y_{ij})$ equals the value of $g(y)$ when $y_{ij} = 1$ minus the value of $g(y)$ when $y_{ij} = 0$, but all other dyads are as in $g(y)$.

This emphasizes that the coefficient θ can be interpreted as the log-odds of an individual tie conditional on all others.

The model terms $g(y)$ are functions of network statistics that we hypothesize may be more or less common than what would be expected in a simple random graph (where all ties have the same probability). For example, specific degree distributions, or triad configurations, or homophily on nodal attributes. We will explore some of these terms in this tutorial, and links to more information are provided in section 3.

One key distinction in model terms is worth keeping in mind: terms are either *dyad independent* or *dyad dependent*. Dyad independent terms (like nodal homophily terms) imply no dependence between dyads—the presence or absence of a tie may depend on nodal attributes, but not on the state of other ties. Dyad dependent terms (like degree terms, or triad terms), by contrast, imply dependence between dyads. Such terms have very different effects, and much of what is different about network models comes from the complex cascading effects that these terms introduce. A model with dyad dependent terms also requires a different estimation algorithm, and you will see some different components in the output.

We'll start by running some simple models to demonstrate the use of the "summary" and "ergm" commands. The ergm package contains several network data sets that we will use for demonstration purposes here.

```
data(package='ergm') # tells us the datasets in our packages
```

Bernoulli model

We begin with the simplest possible model, the Bernoulli or Erdos-Renyi model, which contains only one term to capture the density of the network as a function of a homogenous edge probability. The ergm-term for this is ***edge***. We'll fit this simple model to Padgett's Florentine marriage network. As with all data analysis, we start by looking at our data: using graphical and numerical descriptives.

```
data(florentine) # loads flomarriage and flobusiness data
flomarriage # Let's look at the flomarriage network properties
```

```
 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
```

```
par(mfrow=c(1,2)) # Setup a 2 panel plot (for later)
plot(flomarriage, main="Florentine Marriage", cex.main=0.8) # Plot the flomarriage network
summary(flomarriage~edges) # Look at the $g(y)$ statistic for this model
```

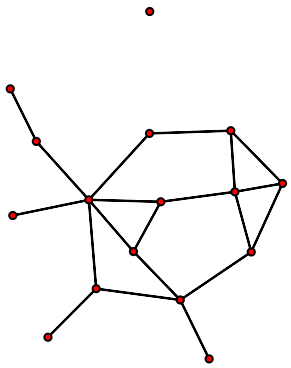
```
edges
   20
```

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

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

```r
summary(flomodel.01) # The fitted model object
```

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

Formula:   flomarriage ~ edges

Iterations:  5 out of 20

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 should we interpret the coefficient from this model? The log-odds of any tie existing 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 always changes the number of ties by 1 for a tie toggled from 0 to 1 (or by -1 for a tie toggled from 1 to 0).

The corresponding probability is:

$$

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

$$

which corresponds to the density we observe in the flomarriage network: there are 20 ties and (16 choose 2 = 16*15/2 =) 120 dyads.

Triad formation

Let's add a term often thought to be a measure of "clustering": the number of completed triangles. The ergm-term for this is ***triangle***. This is a dyad dependent term. As a result, the estimation algorithm automatically changes to MCMC, and because this is a form of stochastic estimation your results may differ slightly.

```
summary(flomarriage~edges+triangle) # Look at the g(y) stats for this model
```

```
   edges triangle
      20        3
```

```
flomodel.02 <- ergm(flomarriage~edges+triangle)
```

```
Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 0.004492
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 0.001591
Step length converged twice. Stopping.
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() f
```

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

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

Formula:   flomarriage ~ edges + triangle

Iterations:  2 out of 20

Monte Carlo MLE Results:
```

```
         Estimate Std. Error MCMC % p-value
edges     -1.6761     0.3485      0  <1e-04 ***
triangle   0.1469     0.5668      0   0.796
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 166.4  on 120  degrees of freedom
 Residual Deviance: 108.1  on 118  degrees of freedom

AIC: 112.1    BIC: 117.6    (Smaller is better.)
```

Now, how should we interpret coefficients?

The conditional log-odds of two actors having a tie is:

$$-1.67 \times \text{change in the number of ties} + 0.14 \times \text{change in number of triangles}$$

- For a tie that will create no triangles, the conditional log-odds is: -1.67.
- if one triangle: $-1.67 + 0.14 = -1.53$
- if two triangles: $-1.67 + 0.14 \times 2 = -1.39$
- the corresponding probabilities are 0.16, 0.18, and 0.20.

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

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

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

```
names(flomodel.02) # the ERGM object contains lots of components.
```

```
 [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] "formula"       "target.stats"  "target.esteq"  "constrained"
[25] "constraints"   "reference"     "estimate"      "offset"
[29] "drop"          "estimable"     "null.lik"      "mle.lik"
```

```
flomodel.02$coef # you can extract/inspect individual components
```

```
     edges   triangle
-1.6760880  0.1468579
```

Nodal covariates: effects on mean degree

We can test whether edge probabilities are a function of wealth. This is a nodal covariate, so we use the ergm-term **nodecov**.

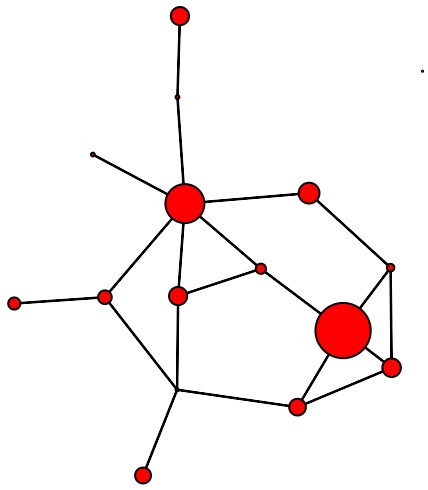
```
wealth <- flomarriage %v% 'wealth' # %v% references vertex attributes
wealth
```

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

```
summary(wealth) # summarize the distribution of wealth
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3.00   17.50   39.00   42.56   48.25  146.00
```

```
plot(flomarriage, vertex.cex=wealth/25, main="Florentine marriage by wealth", cex.main=0.8) # network plot with ver
```




```
summary(flomarriage~edges+nodecov('wealth')) # observed statistics for the model
```

```
         edges nodecov.wealth
            20           2168
```

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

```
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 % 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.

How do we interpret the coefficients here? Note that the wealth effect operates on both nodes in a dyad. The conditional log-odds of a tie between two actors is:

$$-2.59 \times \text{change in the number of ties} + 0.01 \times \text{the wealth of node 1} + 0.01 \times \text{the wealth of node 2}$$

$$-2.59 \times \text{change in the number of ties} + 0.01 \times \text{the sum of the wealth of the two nodes}$$

- for a tie between two nodes with minimum wealth, the conditional log-odds is:
  $-2.59 + 0.01 * (3 + 3) = -2.53$
- for a tie between two nodes with maximum wealth:
  $-2.59 + 0.01 * (146 + 146) = 0.33$
- for a tie between the node with maximum wealth and the node with minimum wealth:
  $-2.59 + 0.01 * (146 + 3) = -1.1$
- The corresponding probabilities are 0.07, 0.58, and 0.25.

Note: This model specification does not include a term for homophily by wealth. It just specifies a relation between wealth and mean degree. To specify homophily on wealth, you would use the ergm-term ***absdiff*** see section 3 below for more information on ergm-terms

Nodal covariates: Homophily

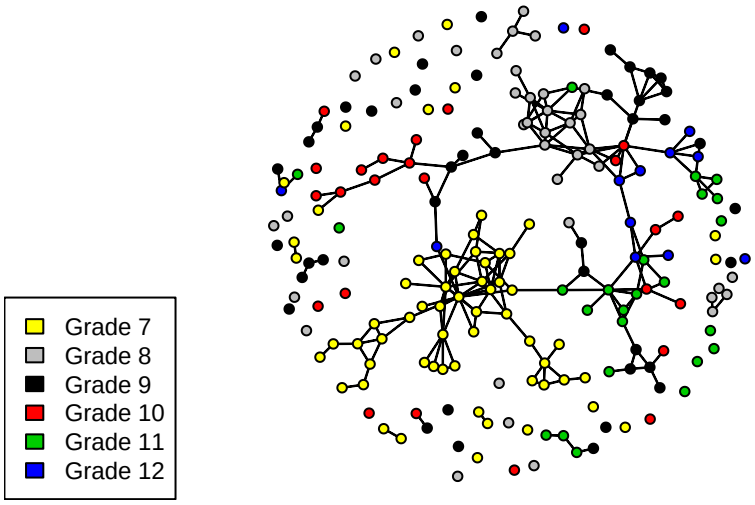
Let's try a larger network, a simulated mutual friendship network based on one of the schools from the Add Health study. Here, we'll examine the homophily in friendships by grade and race. Both are discrete attributes so we use the ergm-term ***nodematch***.

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

```
mesa
```

```
 Network attributes:
  vertices = 205
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 203
    missing edges= 0
    non-missing edges= 203

 Vertex attribute names:
    Grade Race Sex

No edge attributes
```

```
par(mfrow=c(1,1)) # Back to 1-panel plots
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))
```

```
Observed statistic(s) nodematch.Race.Black and nodematch.Race.Other are at their smallest attainable values. Their
Evaluating log-likelihood at the estimate.
```

```
summary(fauxmodel.01)
```

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

Formula:   mesa ~ edges + nodematch("Grade", diff = T) + nodematch("Race",
    diff = T)

Iterations:  8 out of 20

Monte Carlo MLE Results:
                    Estimate Std. Error MCMC % p-value
edges                -6.2328     0.1742      0  <1e-04 ***
nodematch.Grade.7     2.8740     0.1981      0  <1e-04 ***
nodematch.Grade.8     2.8788     0.2391      0  <1e-04 ***
nodematch.Grade.9     2.4642     0.2647      0  <1e-04 ***
nodematch.Grade.10    2.5692     0.3770      0  <1e-04 ***
nodematch.Grade.11    3.2921     0.2978      0  <1e-04 ***
nodematch.Grade.12    3.8376     0.4592      0  <1e-04 ***
nodematch.Race.Black    -Inf     0.0000      0  <1e-04 ***
nodematch.Race.Hisp   0.0679     0.1737      0  0.6959
nodematch.Race.NatAm  0.9817     0.1842      0  <1e-04 ***
```

```
nodematch.Race.Other     -Inf     0.0000      0  <1e-04 ***
nodematch.Race.White   1.2685     0.5371      0  0.0182 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 28987  on 20910  degrees of freedom
 Residual Deviance:  1928  on 20898  degrees of freedom

AIC: 1952    BIC: 2047    (Smaller is better.)

 Warning: The following terms have infinite coefficient estimates:
  nodematch.Race.Black nodematch.Race.Other
```

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
```

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

Note that we would have caught this earlier if we had looked at the $g(y)$ stats at the beginning:

```
summary(mesa ~edges + nodematch('Grade',diff=T) + nodematch('Race',diff=T))
```

```
               edges    nodematch.Grade.7    nodematch.Grade.8
                 203                   75                   33
   nodematch.Grade.9   nodematch.Grade.10   nodematch.Grade.11
                  23                    9                   17
  nodematch.Grade.12 nodematch.Race.Black  nodematch.Race.Hisp
                   6                    0                   53
nodematch.Race.NatAm nodematch.Race.Other nodematch.Race.White
                  46                    0                    4
```

Moral: It's a good idea to check the descriptive statistics of a model in the observed network before fitting the model.

See also the ergm-term ***nodemix*** for fitting mixing patterns other than homophily on discrete nodal attributes.

Directed ties

Let's try a model for a directed network, and examine the tendency for ties to be reciprocated ("mutuality"). The ergm-term for this is ***mutual***. We'll fit this model to the third wave of the classic Sampson Monastery data, and we'll start by taking a look at the network.

```
data(samplk)
ls() # directed data: Sampson's Monks
```

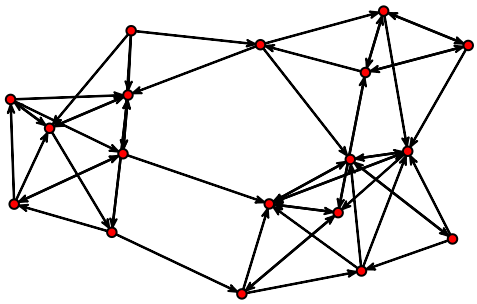
```
 [1] "faux.mesa.high" "fauxmodel.01"   "flobusiness"    "flomarriage"
```

```
 [5] "flomodel.01"    "flomodel.02"    "flomodel.03"    "mesa"
 [9] "samplk1"        "samplk2"        "samplk3"        "wealth"
```

```
samplk3
```

```
 Network attributes:
  vertices = 18
  directed = TRUE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 56
    missing edges= 0
    non-missing edges= 56

 Vertex attribute names:
    cloisterville group vertex.names

No edge attributes
```

```
plot(samplk3)
```



```
summary(samplk3~edges+mutual)
```

```
 edges mutual
    56     15
```

The plot now shows the direction of a tie, and the $g(y)$ statistics for this model in this network are 56 total ties, and 15 mutual dyads (so 30 of the 56 ties are mutual ties).

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

```
Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 0.001104
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 0.0006444
Step length converged twice. Stopping.
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() f
```

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

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

Formula:   samplk3 ~ edges + mutual

Iterations:  2 out of 20

Monte Carlo MLE Results:
       Estimate Std. Error MCMC % p-value
edges   -2.1569     0.2158      0  <1e-04 ***
mutual   2.2997     0.4811      0  <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 424.2  on 306  degrees of freedom
 Residual Deviance: 268.0  on 304  degrees of freedom

AIC: 272    BIC: 279.4    (Smaller is better.)
```

There is a strong and significant mutuality effect. The coefficients for the edges and mutual terms roughly cancel for a mutual tie, so the conditional odds of a mutual tie are about even, and the probability is about 50%. By contrast a non-mutual tie has a conditional log-odds of -2.16, or 10% probability.

Triangle terms in directed networks can have many different configurations, given the directional ties. Many of these configurations are coded up as ergm-terms (and we'll talk about these more below).

Missing data

It is important to distinguish between the absence of a tie, and the absence of data on whether a tie exists. You should not code both of these as "0". The *ergm* package recognizes and handles missing data appropriately, as long as you identify the data as missing. Let's explore this with a simple example.

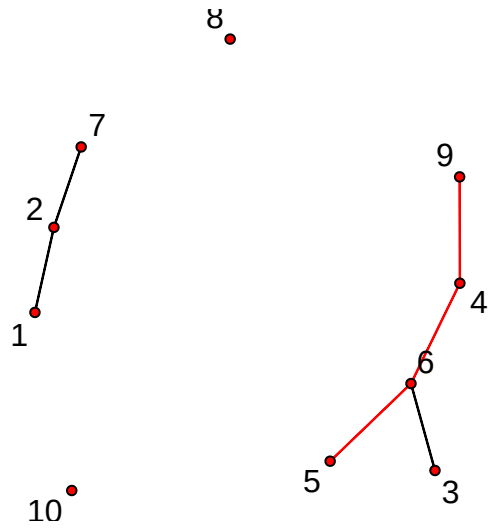
Let's start with estimating an ergm on a network with two missing ties, where both ties are identified as missing.

```
missnet <- network.initialize(10,directed=F)
missnet[1,2] <- missnet[2,7] <- missnet[3,6] <- 1
missnet[4,6] <- missnet[4,9] <- missnet[5,6] <- NA
summary(missnet)
```

```
Network attributes:
  vertices = 10
  directed = FALSE
  hyper = FALSE
```

```
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
 total edges = 6
   missing edges = 3
   non-missing edges = 3
 density = 0.06666667

Vertex attributes:
  vertex.names:
   character valued attribute
   10 valid vertex names

No edge attributes

Network adjacency matrix:
   1 2 3  4  5  6 7 8  9 10
1  0 1 0  0  0  0 0 0  0  0
2  1 0 0  0  0  0 1 0  0  0
3  0 0 0  0  0  1 0 0  0  0
4  0 0 0  0  0 NA 0 0 NA  0
5  0 0 0  0  0 NA 0 0  0  0
6  0 0 1 NA NA  0 0 0  0  0
7  0 1 0  0  0  0 0 0  0  0
8  0 0 0  0  0  0 0 0  0  0
9  0 0 0 NA  0  0 0 0  0  0
10 0 0 0  0  0  0 0 0  0  0
```

```r
# plot missnet with missing edge colored red.
tempnet <- missnet
tempnet[4,6] <- tempnet[4,9] <- tempnet[5,6] <- 1
missnetmat <- as.matrix(missnet)
missnetmat[is.na(missnetmat)] <- 2
plot(tempnet,label = network.vertex.names(tempnet),edge.col = missnetmat)
```




```
summary(missnet~edges)
```

```
edges
    3
```

```
summary(ergm(missnet~edges))
```

```
Evaluating log-likelihood at the estimate.


==========================
Summary of model fit
==========================

Formula:   missnet ~ edges

Iterations:  5 out of 20

Monte Carlo MLE Results:
      Estimate Std. Error MCMC %  p-value
edges  -2.5649     0.5991      0 0.000109 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 58.22  on 42  degrees of freedom
 Residual Deviance: 21.61  on 41  degrees of freedom

AIC: 23.61    BIC: 25.35    (Smaller is better.)
```

The coefficient equals -2.56, which corresponds to a probability of 7.14%. Our network has 3 ties, out of the 42

non-missing nodal pairs (10 choose 2 minus 3): $3/42 = 7.14\%$. So our estimate represents the probability of a tie in the observed sample.

Now let's assign those missing ties the value "0" and see what happens.

```
missnet_bad <- missnet
missnet_bad[4,6] <- missnet_bad[4,9] <- missnet_bad[5,6] <- 0
summary(missnet_bad)
```

```
Network attributes:
  vertices = 10
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
 total edges = 3
   missing edges = 0
   non-missing edges = 3
 density = 0.06666667

Vertex attributes:
  vertex.names:
   character valued attribute
   10 valid vertex names

No edge attributes

Network adjacency matrix:
   1 2 3 4 5 6 7 8 9 10
1  0 1 0 0 0 0 0 0 0  0
2  1 0 0 0 0 0 1 0 0  0
3  0 0 0 0 0 1 0 0 0  0
4  0 0 0 0 0 0 0 0 0  0
5  0 0 0 0 0 0 0 0 0  0
6  0 0 1 0 0 0 0 0 0  0
7  0 1 0 0 0 0 0 0 0  0
8  0 0 0 0 0 0 0 0 0  0
9  0 0 0 0 0 0 0 0 0  0
10 0 0 0 0 0 0 0 0 0  0
```

```
summary(ergm(missnet_bad~edges))
```

```
Evaluating log-likelihood at the estimate.


==========================
Summary of model fit
==========================

Formula:   missnet_bad ~ edges

Iterations:  5 out of 20

Monte Carlo MLE Results:
      Estimate Std. Error MCMC % p-value
edges  -2.6391     0.5976      0  <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 62.38  on 45  degrees of freedom
```

```
 Residual Deviance: 22.04  on 44  degrees of freedom

AIC: 24.04    BIC: 25.85    (Smaller is better.)
```

The coefficient is smaller now because the missing ties are counted as "0", and translates to a conditional tie probability of 6.67%. It's a small difference in this case (and a small network, with little missing data).

MORAL: If you have missing data on ties, be sure to identify them by assigning the "NA" code. This is particularly important if you're reading in data as an edgelist, as all dyads without edges are implicitly set to "0" in this case.

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

Many ERGM terms are simple counts of configurations (e.g., edges, nodal degrees, stars, triangles), but others are more complex functions of these configurations (e.g., geometrically weighted degrees and shared partners). In theory, any configuration (or function of configurations) can be a term in an ERGM. In practice, however, these terms have to be constructed before they can be used—that is, one has to explicitly write an algorithm that defines and calculates the network statistic of interest. This is another key way that ERGMs differ from traditional linear and general linear models.

The terms that can be used in a model also depend on the type of network being analyzed: directed or undirected, one-mode or two-mode ("bipartite"), binary or valued edges.

Terms provided with ergm

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

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

A table of commonly used terms can be found here

A more complete discussion of many of these terms can be found in the 'Specifications' paper in the *Journal of Statistical Software v24(4)*

Finally, note that models with only dyad independent terms are estimated in statnet using a logistic regression algorithm to maximize the likelihood. Dyad dependent terms require a different approach to estimation, which, in statnet, is based on a Monte Carlo Markov Chain (MCMC) algorithm that stochastically approximates the Maximum Likelihood.

Coding new ergm-terms

We have recently released a new package (`ergm.userterms`) that makes it much easier to write one's own ergm-terms. The package is available on CRAN, and installing it will include the tutorial (ergmuserterms.pdf). Alternatively, the tutorial can be found in the *Journal of Statistical Software 52(2)*, and some introductory slides from the workshop we teach on coding ergm-terms can be found here.

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, the many environments for building R and the rapid changes in these environments make these instructions obsolete quickly).

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: ~.
Parameters:
         edges nodecov.wealth
   -2.59492903     0.01054591

Stored network statistics:
      edges nodecov.wealth
 [1,]    22           2305
 [2,]    21           2053
 [3,]    27           3233
 [4,]    26           2994
 [5,]    18           1763
 [6,]    14           1874
 [7,]    23           2443
 [8,]    18           1646
 [9,]    18           2007
[10,]    22           1953
```

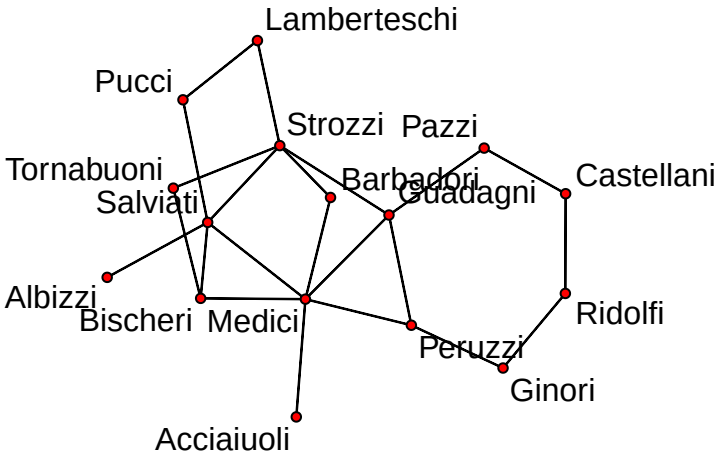
```
length(flomodel.03.sim)
```

```
[1] 10
```

```
flomodel.03.sim[[1]]
```

```
 Network attributes:
  vertices = 16
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 22
    missing edges= 0
    non-missing edges= 22

 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.

Simulation can be used for many purposes: to examine the range of variation that could be expected from this model, both in the sufficient statistics that define the model, and in other statistics not explicitly specified by the model. Simulation will play a large role in analyzing egocentrically sampled data in section 7 below. And if you take the `tergm` workshop, you will see how we can use simulation to examine the temporal implications of a model based on a single cross-sectional egocentrically sampled dataset.

For now, we will examine one of the primary uses of simulation in the ergm package: using simulated data from the model to evaluate goodness of fit to the observed data.

5. Examining the quality of model fit – GOF

ERGMs can be seen as generative models when they represent the process that governs the global patterns of tie prevalence from a local perspective: the perspective of the nodes involved in the particular micro-configurations represented by the ergm-terms in the model. The locally generated processes in turn aggregate up to produce characteristic global network properties, even though these global properties are not explicit terms in the model.

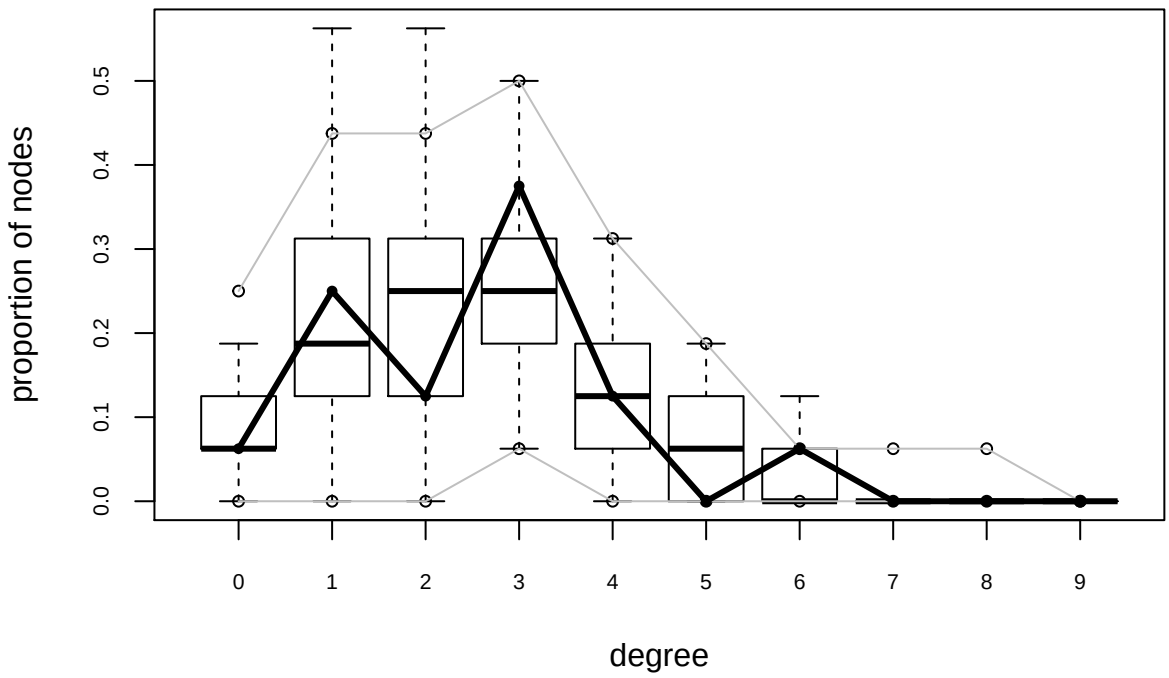
One test of whether a local model "fits the data" is therefore how well it reproduces the observed global network properties *that are not in the model*. We do this by choosing a network statistic that is not in the model, and comparing the value of this statistic observed in the original network to the distribution of values we get in simulated networks from our model, using the **gof** function.

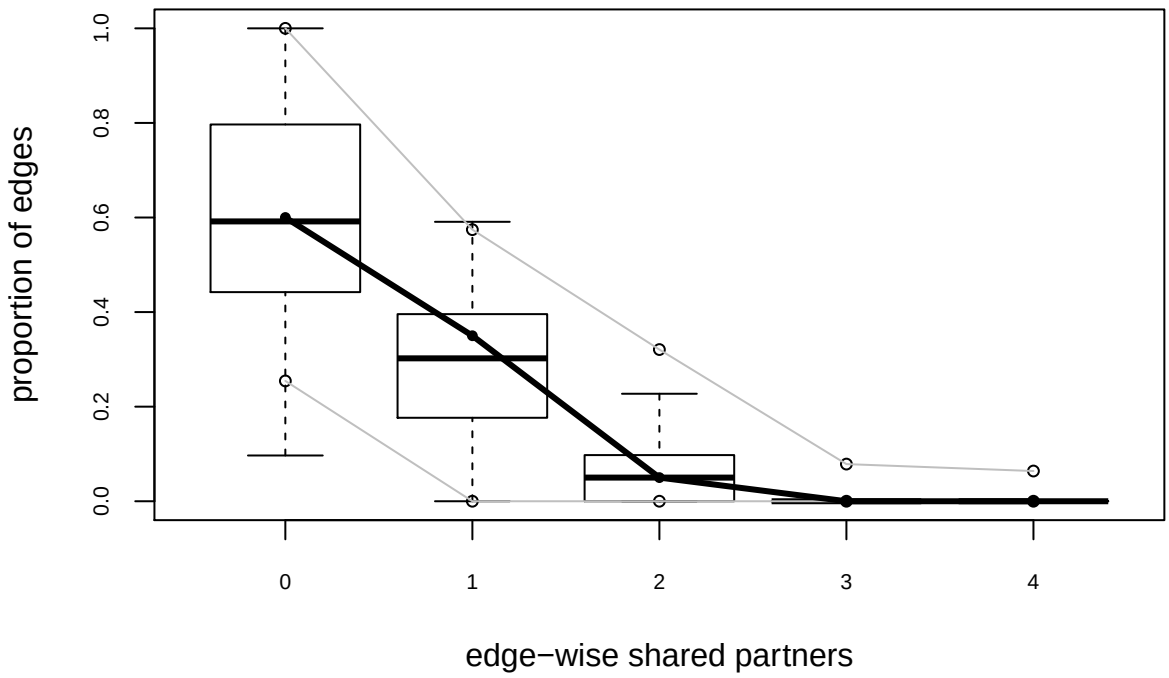
The **gof** function is a bit different than the **summary**, **ergm**, and **simulate** functions, in that it currently only takes 3 ergm-terms as arguments: degree, esp (edgwise share partners), and distance (geodesic distances). Each of these terms captures an aggregate network distribution, at either the node level (degree), the edge level (esp), or the dyad level (distance).

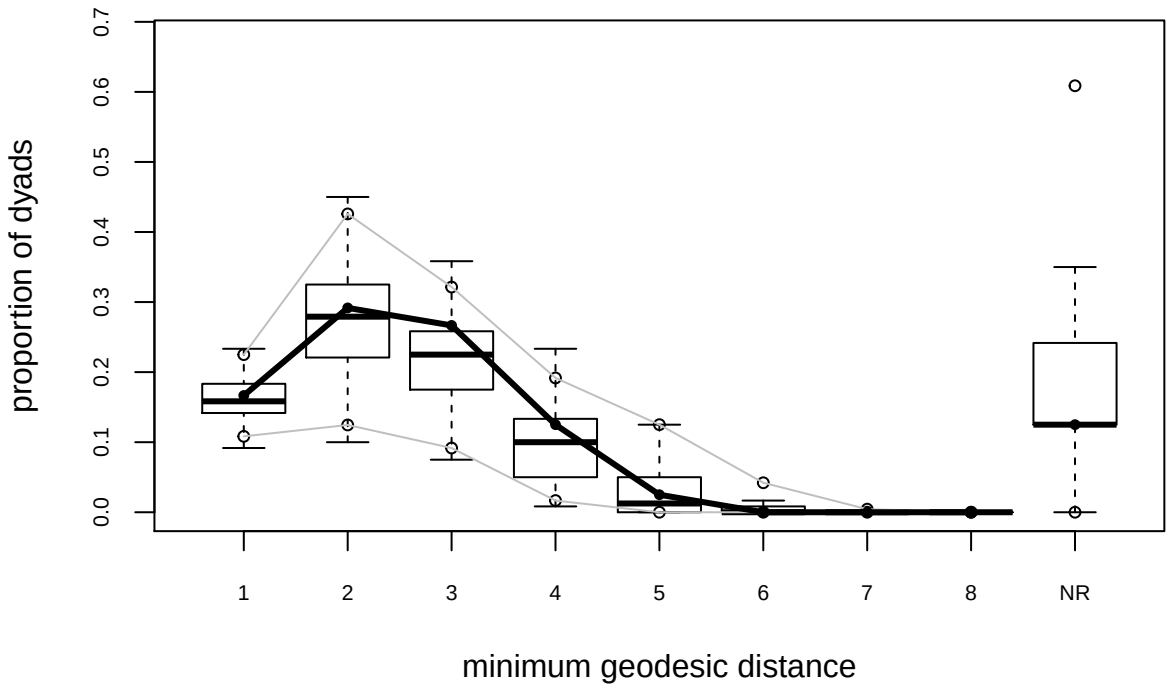
```
flomodel.03.gof <- gof(flomodel.03~degree + esp + distance)
flomodel.03.gof
```

```
Goodness-of-fit for degree

  obs min mean max MC p-value
0   1   0 1.39   5       1.00
1   4   0 3.46   9       0.94
2   2   0 3.88   9       0.52
3   6   1 3.71   9       0.24
4   2   0 1.84   5       1.00
5   0   0 0.98   3       0.72
6   1   0 0.42   2       0.80
7   0   0 0.21   1       1.00
8   0   0 0.11   2       1.00

Goodness-of-fit for edgewise shared partner

     obs min  mean max MC p-value
esp0  12   3 11.92  21       1.00
esp1   7   0  5.97  16       0.88
esp2   1   0  1.54   9       1.00
esp3   0   0  0.22   3       1.00
esp4   0   0  0.08   2       1.00
esp5   0   0  0.01   1       1.00

Goodness-of-fit for minimum geodesic distance

    obs min  mean max MC p-value
1    20  11 19.74  31       0.96
2    35  12 33.03  54       0.90
3    32   9 26.01  43       0.48
4    15   1 11.81  28       0.78
5     3   0  3.56  18       0.94
6     0   0  0.84  10       1.00
7     0   0  0.07   5       1.00
8     0   0  0.02   2       1.00
Inf  15   0 24.92  81       1.00
```

```
plot(flomodel.03.gof)
```


proportion of nodes
0.0
0.1
0.2
0.3
0.4
0.5
0
1
2
3
4
5
6
7
8
9
degree

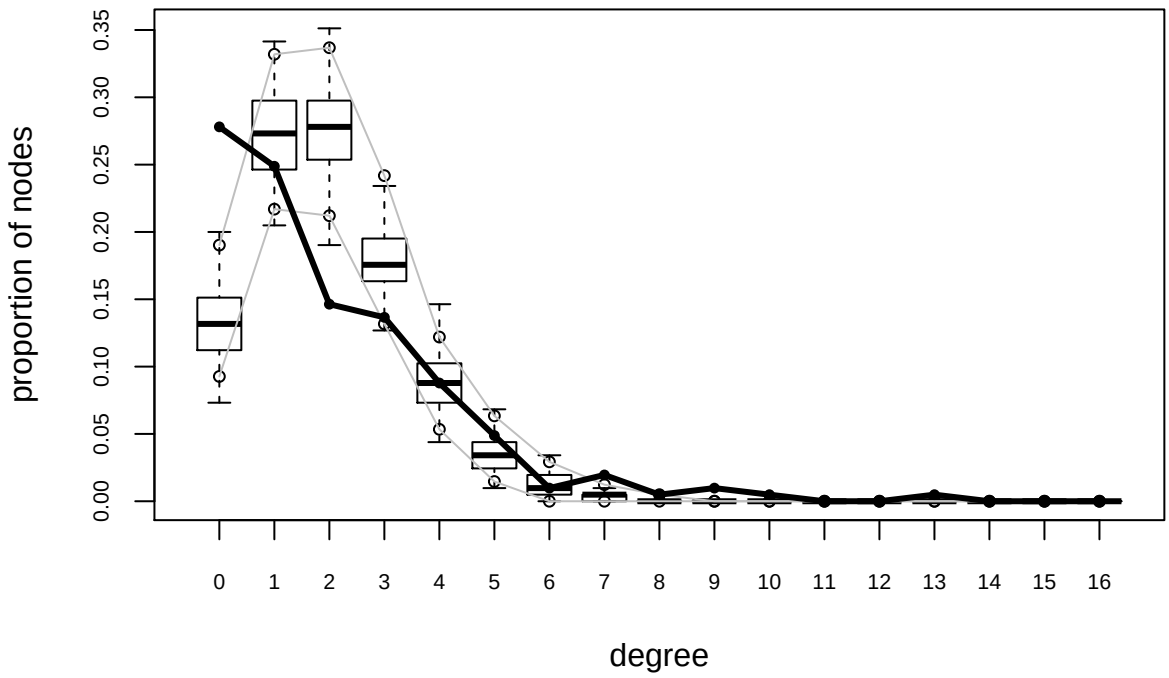

proportion of edges
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
edge−wise shared partners

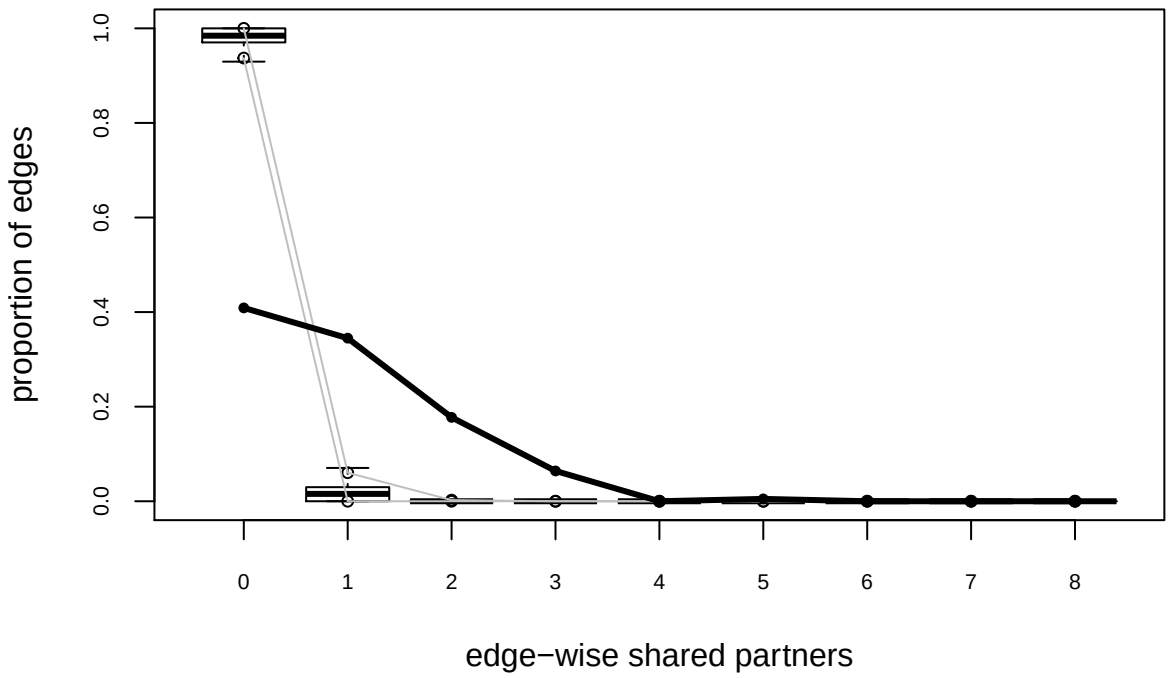


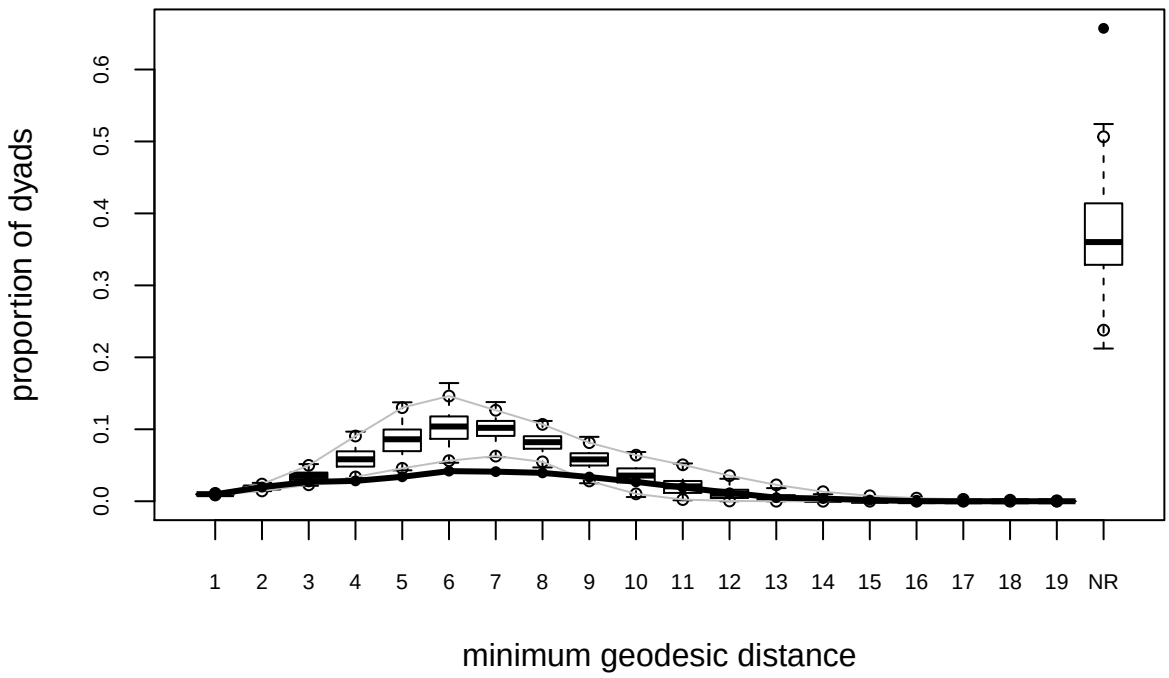

```
mesamodel.02 <- ergm(mesa~edges)
```

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

```
mesamodel.02.gof <- gof(mesamodel.02~degree + esp + distance,
                        control.gof.formula(nsim=10))
plot(mesamodel.02.gof)
```


proportion of nodes
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
degree


proportion of edges
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7
8
edge−wise shared partners



For a good example of model exploration and fitting for the Add Health Friendship networks, see Goodreau, Kitts & Morris, *Demography* 2009.
For more technical details on the approach, see Hunter, Goodreau and Handcock *JASA* 2008

6. Diagnostics: troubleshooting and checking for model degeneracy

The computational algorithms in `ergm` use MCMC to estimate the likelihood function when dyad dependent terms are in the model. Part of this process involves simulating a set of networks to use as a sample for approximating the unknown component of the likelihood: the $k(\theta)$ term in the denominator.

When a model is not a good representation of the observed network, these simulated networks may be far enough away from the observed network that the estimation process is affected. In the worst case scenario, the simulated networks will be so different that the algorithm fails altogether.

For more detailed discussion of model degeneracy in the ERGM context, see the papers by Mark Handcock referenced below.

In the worst case scenario, we end up not being able to obtain coefficent estimates, so we can't use the GOF function to identify how the model simulations deviate from the observed data. In this case, however, we can use the MCMC diagnostics to observe what is happening with the simulation algorithm, and this (plus some experience and intuition about the behavior of ergm-terms) can help us improve the model specification.

Below we show a simple example of a model that converges, and one that doesn't, and how to use the MCMC diagnostics to improve a model that isn't converging.

What it looks like when a model converges properly

We will first consider a simulation where the algorithm works using the program defaults, and observe the behavior of the MCMC estimation algorithm using the `mcmc.diagnostics` function.

```
summary(flobusiness~edges+degree(1))
```

```
  edges degree1
     15       3
```

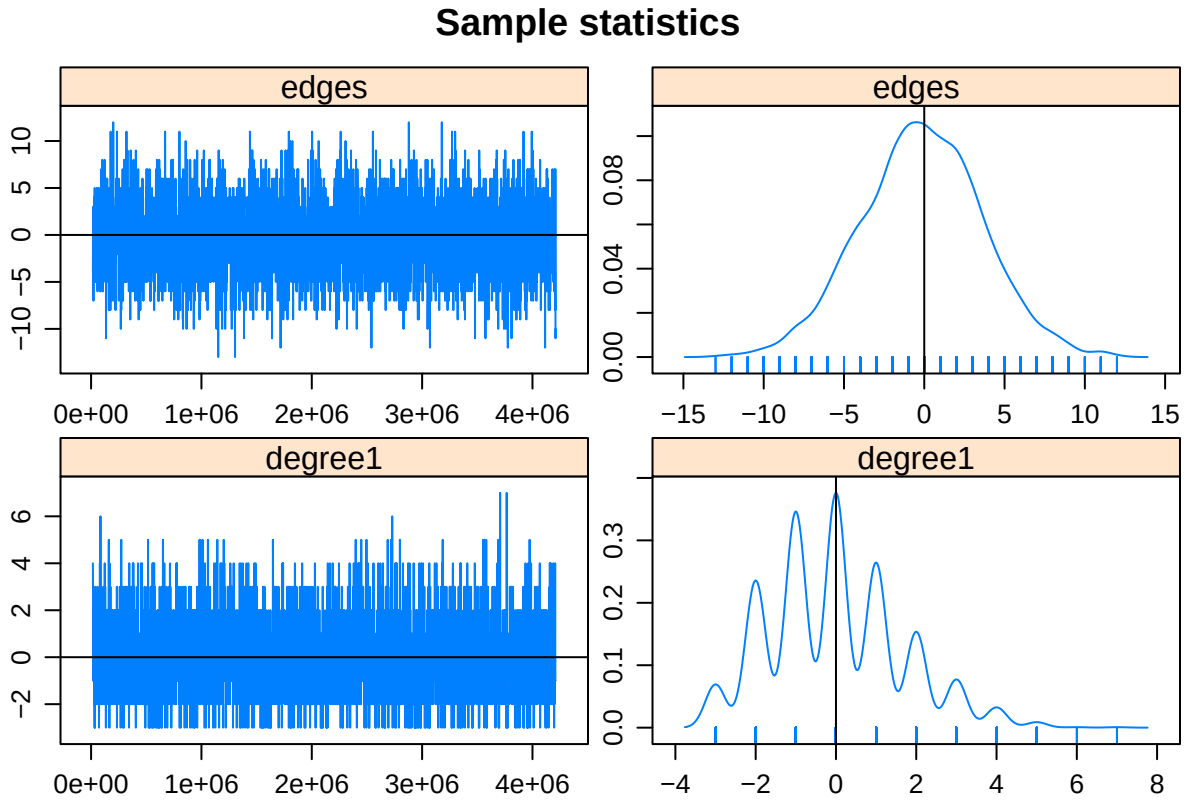
```
fit <- ergm(flobusiness~edges+degree(1))
```

```
Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 0.2587
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 0.001645
Step length converged twice. Stopping.
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() f
```

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

```
Sample statistics summary:

Iterations = 16384:4209664
Thinning interval = 1024
Number of chains = 1
Sample size per chain = 4096

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

            Mean    SD Naive SE Time-series SE
edges   -0.16235 3.776  0.05900        0.05721
degree1 -0.02441 1.656  0.02587        0.02587

2. Quantiles for each variable:

        2.5% 25% 50% 75% 97.5%
edges     -8  -3   0   2     7
degree1   -3  -1   0   1     4


Sample statistics cross-correlations:
             edges    degree1
edges    1.0000000 -0.4469011
degree1 -0.4469011  1.0000000

Sample statistics auto-correlation:
Chain 1
                edges      degree1
Lag 0     1.000000000  1.000000000
Lag 1024 -0.030943830 -0.006906477
Lag 2048  0.006677487  0.025075410
Lag 3072 -0.006159053  0.002537837
Lag 4096 -0.005477645 -0.002805939
Lag 5120 -0.011692893 -0.016094690

Sample statistics burn-in diagnostic (Geweke):
Chain 1

Fraction in 1st window = 0.1
```

```
Fraction in 2nd window = 0.5

  edges degree1
  0.964   1.476

Individual P-values (lower = worse):
    edges   degree1
0.3350686 0.1398374
Joint P-value (lower = worse):  0.0800707 .

Loading required namespace: latticeExtra

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




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

This is what you want to see in the MCMC diagnostics: the MCMC sample statistics are varying randomly around the observed values at each step (so the chain is "mixing" well) and the difference between the observed and simulated values of the sample statistics have a roughly bell-shaped distribution, centered at 0. The sawtooth pattern visible on the degree term deviation plot is due to the combination of discrete values and small range in the statistics: the observed number of degree 1 nodes is 3, and only a few discrete values are produced by the simulations. So the sawtooth pattern is is an inherent property of the statistic, not a problem with the fit.

There are many control parameters for the MCMC algorithm ("help(control.ergm)"), and we'll play with some of these below. To see what the algorithm is doing at each step, you can drop the sampling interval down to 1:

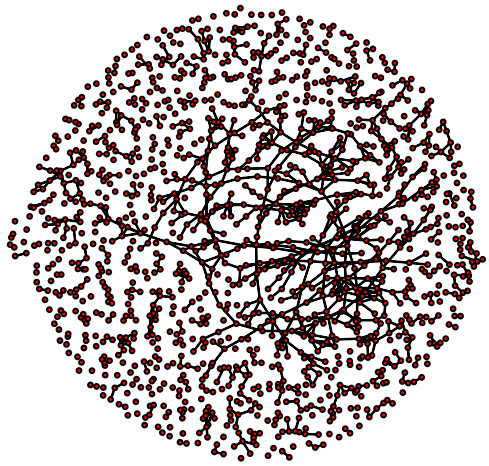
```
fit <- ergm(flobusiness~edges+degree(1),
control=control.ergm(MCMC.interval=1)
```

This runs a version with every network returned, and might be useful if you are trying to debug a bad model fit.

What it looks like when a model fails

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

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



```
summary(magnolia~edges+triangle)
```

```
   edges triangle
     974      169
```

```
fit <- ergm(magnolia~edges+triangle)
```

```
Iteration 1 of at most 20:
Convergence test P-value: 1.4e-87
The log-likelihood improved by 1.183
Iteration 2 of at most 20:
Convergence test P-value: 3.8e-04
The log-likelihood improved by 0.1518
Iteration 3 of at most 20:
```

```
Error: Number of edges in a simulated network exceeds that in the observed by a factor of more than 20. This is a s
```

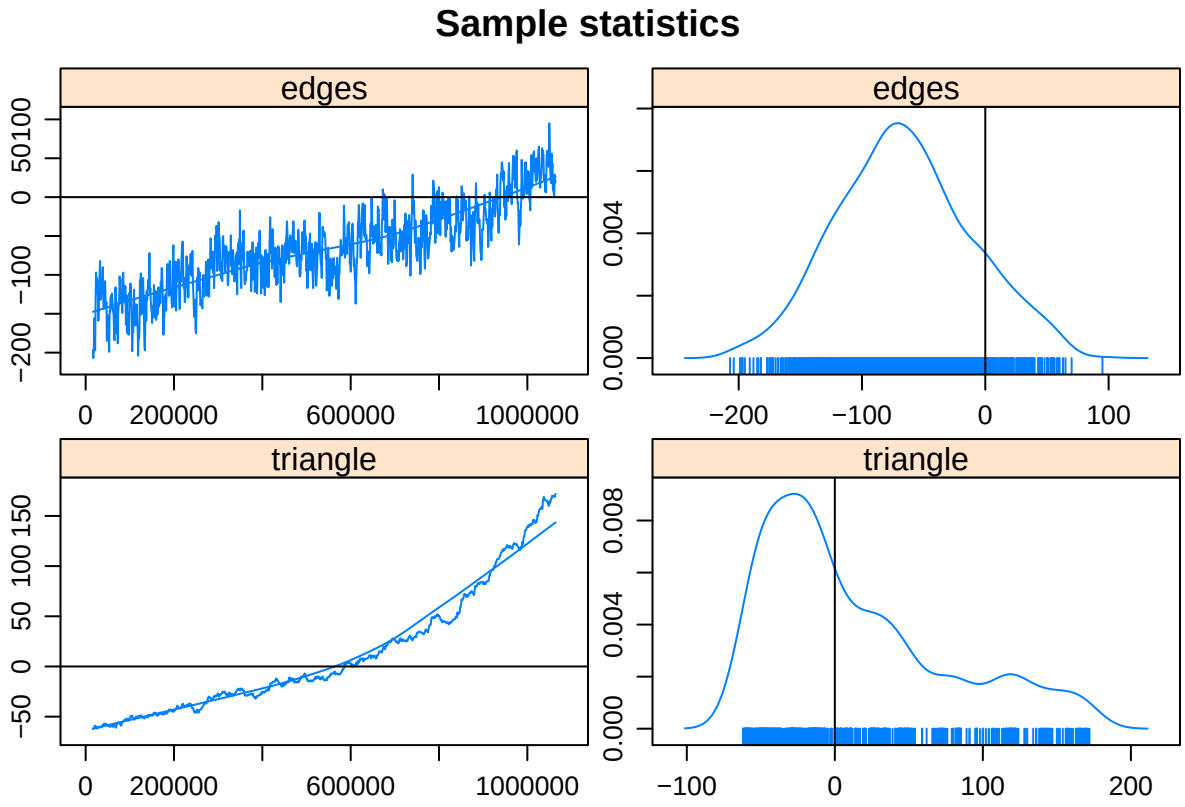
Very interesting. In the process of trying to fit this model, the algorithm heads off into networks that are much much more dense than the observed network. This is such a clear indicator of a degenerate model specification that the algorithm stops after 3 iterations, to avoid heading off into areas that would cause memory issues. If you'd like to peek a bit more under the hood, you can stop the algorithm earlier to catch where it's heading:

```
fit <- ergm(magnolia~edges+triangle, control=control.ergm(MCMLE.maxit=2))
```

```
Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 2:
The log-likelihood improved by 8.688
Iteration 2 of at most 2:
The log-likelihood improved by 3.539
Step length converged once. Increasing MCMC sample size.
MCMLE estimation did not converge after 2 iterations. The estimated coefficients may not be accurate. Estimation ma
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() f
```

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

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



Clearly, somewhere very bad.

How about trying the more robust version of modeling triangles: the geometrically-weighed edgewise shared partner term (GWESP)? (For a technical introduction to GWESP see Hunter and Handcock, 2006; for a more intuitive description and empirical application see Goodreau, Kitts & Morris, 2009)

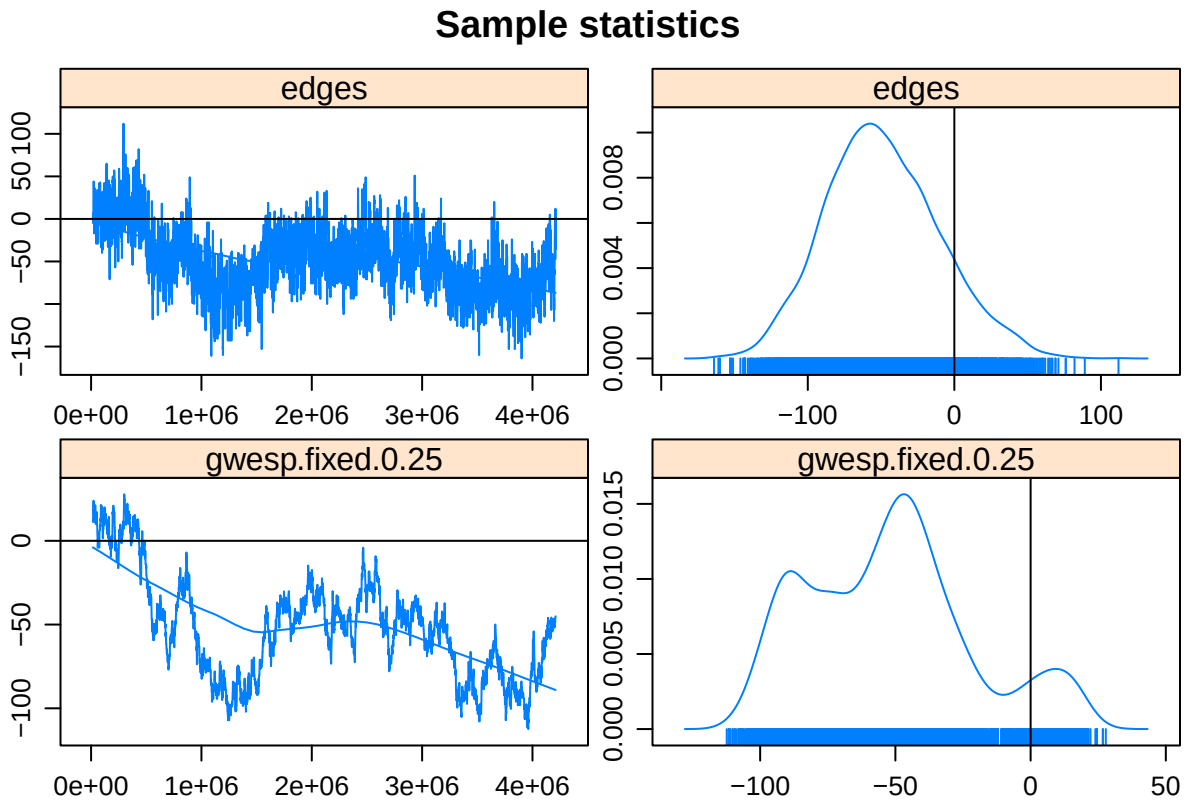
```
fit <- ergm(magnolia~edges+gwesp(0.25,fixed=T),verbose=T)
```

```
Evaluating network in model
Initializing Metropolis-Hastings proposal(s): ergm:MH_TNT
Initializing model.
Using initial method 'MPLE'.
```

```
Fitting initial model.
MPLE covariate matrix has 85 rows.
Fitting ERGM.
Starting maximum likelihood estimation via 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
       -7.350244         2.147119
Sampler accepted  67.306% of 1048576 proposed steps.
Sample size = 1024 by 1024
Back from unconstrained MCMC. Average statistics:
           edges gwesp.fixed.0.25
        44.18457        -37.64275
Average estimating equation values:
           edges gwesp.fixed.0.25
        44.18457        -37.64275
is.inCH: iter= 1, outside hull.
iter= 1, est=0.797856, low=0.000000, high=1.000000, test=0.
is.inCH: iter= 1, outside hull.
iter= 2, est=0.655364, low=0.000000, high=0.797856, test=0.
is.inCH: iter= 1, inside hull.
iter= 3, est=0.734155, low=0.655364, high=0.797856, test=1.
is.inCH: iter= 1, inside hull.
iter= 4, est=0.767319, low=0.734155, high=0.797856, test=1.
is.inCH: iter= 1, outside hull.
iter= 5, est=0.751117, low=0.734155, high=0.767319, test=0.
is.inCH: iter= 1, inside hull.
iter= 6, est=0.759306, low=0.751117, high=0.767319, test=1.
is.inCH: iter= 1, inside hull.
iter= 7, est=0.763333, low=0.759306, high=0.767319, test=1.
is.inCH: iter= 1, outside hull.
iter= 8, est=0.761325, low=0.759306, high=0.763333, test=0.
is.inCH: iter= 1, outside hull.
iter= 9, est=0.760317, low=0.759306, high=0.761325, test=0.
is.inCH: iter= 1, inside hull.
iter= 10, est=0.760821, low=0.760317, high=0.761325, test=1.
is.inCH: iter= 1, inside hull.
iter= 11, est=0.761073, low=0.760821, high=0.761325, test=1.
Calling MCMLE Optimization...
Using Newton-Raphson Step with step length  0.761073059999765  ...
Using lognormal metric (see control.ergm function).
Using log-normal approx (no optim)
The log-likelihood improved by 3.857
Iteration 2 of at most 20 with parameter:
           edges gwesp.fixed.0.25
       -7.434010         2.318083
Sampler accepted  62.746% of 1048576 proposed steps.
Sample size = 1024 by 1024
Back from unconstrained MCMC. Average statistics:
           edges gwesp.fixed.0.25
       26.990234         5.689416
Average estimating equation values:
           edges gwesp.fixed.0.25
       26.990234         5.689416
is.inCH: iter= 1, inside hull.
iter= 1, est=1.000000, low=1.000000, high=1.000000, test=1.
Calling MCMLE Optimization...
Using Newton-Raphson Step with step length  1  ...
Using lognormal metric (see control.ergm function).
```

```
Using log-normal approx (no optim)
The log-likelihood improved by 0.3968
Iteration 3 of at most 20 with parameter:
           edges gwesp.fixed.0.25
       -7.469402         2.346484
Sampler accepted  60.057% of 1048576 proposed steps.
Sample size = 1024 by 1024
Back from unconstrained MCMC. Average statistics:
           edges gwesp.fixed.0.25
        40.78906         35.24110
Average estimating equation values:
           edges gwesp.fixed.0.25
        40.78906         35.24110
is.inCH: iter= 1, inside hull.
iter= 1, est=1.000000, low=1.000000, high=1.000000, test=1.
Calling MCMLE Optimization...
Using Newton-Raphson Step with step length  1  ...
Using lognormal metric (see control.ergm function).
Using log-normal approx (no optim)
The log-likelihood improved by 1.485
Step length converged once. Increasing MCMC sample size.
Iteration 4 of at most 20 with parameter:
           edges gwesp.fixed.0.25
       -7.474025         2.267559
Sampler accepted  65.515% of 4194304 proposed steps.
Sample size = 4096 by 4096
Back from unconstrained MCMC. Average statistics:
           edges gwesp.fixed.0.25
       -48.89136        -52.64497
Average estimating equation values:
           edges gwesp.fixed.0.25
       -48.89136        -52.64497
is.inCH: iter= 1, inside hull.
iter= 1, est=1.000000, low=1.000000, high=1.000000, test=1.
Calling MCMLE Optimization...
Using Newton-Raphson Step 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 1.499
Step length converged twice. Stopping.
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() f
mcmc.diagnostics(fit)

Sample statistics summary:

Iterations = 16384:4209664
Thinning interval = 1024
Number of chains = 1
Sample size per chain = 4096

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

                  Mean    SD Naive SE Time-series SE
edges            -48.89 38.82   0.6066          7.866
```

```
gwesp.fixed.0.25 -52.64 30.47   0.4760         10.052

2. Quantiles for each variable:

                   2.5%    25%    50%   75% 97.5%
edges            -119.0 -76.00 -52.00 -24.0 33.00
gwesp.fixed.0.25 -100.7 -77.08 -52.61 -36.6 14.77


Sample statistics cross-correlations:
                     edges gwesp.fixed.0.25
edges            1.0000000        0.7684542
gwesp.fixed.0.25 0.7684542        1.0000000

Sample statistics auto-correlation:
Chain 1
             edges gwesp.fixed.0.25
Lag 0    1.0000000        1.0000000
Lag 1024 0.8267218        0.9955233
Lag 2048 0.7310688        0.9911892
Lag 3072 0.6717042        0.9870599
Lag 4096 0.6346548        0.9829636
Lag 5120 0.6102523        0.9790446

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
           9.569            8.033

Individual P-values (lower = worse):
           edges gwesp.fixed.0.25
    1.080726e-21     9.484223e-16
Joint P-value (lower = worse):  0.001516495 .

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




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

Better, but not great. Here we'll change two things – add some more reasonable terms to the model, and add in some robustness to the fitting algorithm by making the MCMC simulation longer.

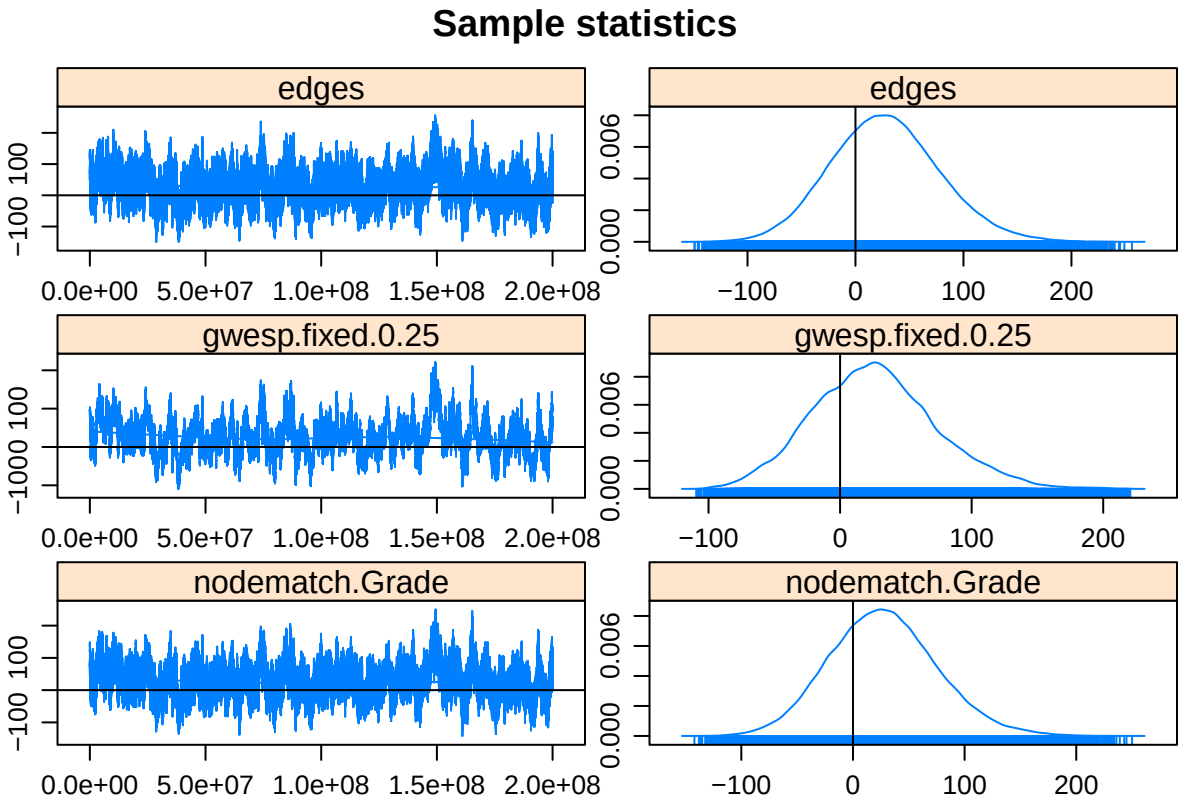
```
fit <- ergm(magnolia~edges+gwesp(0.25,fixed=T)+nodematch('Grade')+
              nodematch('Race')+nodematch('Sex'),
            control = control.ergm(MCMC.samplesize=50000,MCMC.interval=1000),
            verbose=T)
```

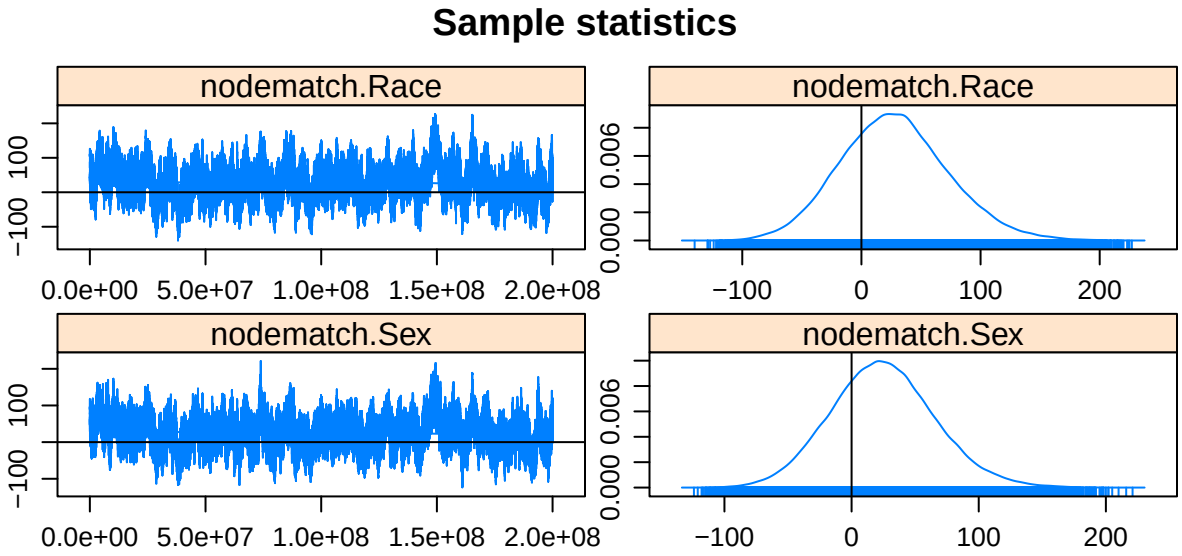
```
Evaluating network in model
Initializing Metropolis-Hastings proposal(s): ergm:MH_TNT
Initializing model.
Using initial method 'MPLE'.
Fitting initial model.
MPLE covariate matrix has 211 rows.
Fitting ERGM.
Starting maximum likelihood estimation via MCMLE:
Density guard set to 19563 from an initial 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
Sampler accepted  30.444% of 50000000 proposed steps.
Sample size = 50000 by 50000
Back from unconstrained MCMC. Average statistics:
           edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
       104.50542         42.95520        102.77140        100.89348
```

```
   nodematch.Sex
        87.45524
Average estimating equation values:
           edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
       104.50542         42.95520        102.77140        100.89348
   nodematch.Sex
        87.45524
is.inCH: iter= 1, inside hull.
iter= 1, est=1.000000, low=1.000000, high=1.000000, test=1.
Calling MCMLE Optimization...
Using Newton-Raphson Step with step length  1  ...
Using lognormal metric (see control.ergm function).
Using log-normal approx (no optim)
The log-likelihood improved by 4.943
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20 with parameter:
           edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
      -9.7991219        1.7993887        2.7523455        0.9231004
   nodematch.Sex
       0.7793418
Sampler accepted  31.353% of 200000000 proposed steps.
Sample size = 200000 by 2e+05
Back from unconstrained MCMC. Average statistics:
           edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
        27.78893         26.07290         27.23695         27.70463
   nodematch.Sex
        24.74043
Average estimating equation values:
           edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
        27.78893         26.07290         27.23694         27.70463
   nodematch.Sex
        24.74042
is.inCH: iter= 1, inside hull.
iter= 1, est=1.000000, low=1.000000, high=1.000000, test=1.
Calling MCMLE Optimization...
Using Newton-Raphson Step 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 0.2216
Step length converged twice. Stopping.
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() f
mcmc.diagnostics(fit)

Sample statistics summary:

Iterations = 16000:200015000
Thinning interval = 1000
Number of chains = 1
Sample size per chain = 2e+05

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

                  Mean    SD Naive SE Time-series SE
edges            27.79 50.41  0.11273          2.739
```

```
gwesp.fixed.0.25 26.07 46.95  0.10499          3.843
nodematch.Grade  27.24 48.16  0.10769          2.853
nodematch.Race   27.70 45.63  0.10204          2.561
nodematch.Sex    24.74 41.14  0.09199          2.165

2. Quantiles for each variable:

                   2.5%    25%   50%   75% 97.5%
edges            -67.00 -7.000 26.00 60.00 132.0
gwesp.fixed.0.25 -60.46 -6.689 24.29 55.41 125.5
nodematch.Grade  -62.00 -6.000 26.00 58.00 127.0
nodematch.Race   -57.00 -4.000 26.00 57.00 123.0
nodematch.Sex    -53.00 -3.000 24.00 51.00 110.0


Sample statistics cross-correlations:
                     edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
edges            1.0000000        0.8714065       0.9665097      0.9519157
gwesp.fixed.0.25 0.8714065        1.0000000       0.8889242      0.8644635
nodematch.Grade  0.9665097        0.8889242       1.0000000      0.9260056
nodematch.Race   0.9519157        0.8644635       0.9260056      1.0000000
nodematch.Sex    0.9165936        0.8206331       0.8894860      0.8743166
                 nodematch.Sex
edges                0.9165936
gwesp.fixed.0.25     0.8206331
nodematch.Grade      0.8894860
nodematch.Race       0.8743166
nodematch.Sex        1.0000000

Sample statistics auto-correlation:
Chain 1
             edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
Lag 0    1.0000000        1.0000000       1.0000000      1.0000000
Lag 1000 0.9479083        0.9980284       0.9674246      0.9579362
Lag 2000 0.9103742        0.9961005       0.9413142      0.9267702
Lag 3000 0.8825336        0.9941959       0.9200072      0.9030441
Lag 4000 0.8614533        0.9923221       0.9025121      0.8844598
Lag 5000 0.8446617        0.9904794       0.8878142      0.8693348
         nodematch.Sex
Lag 0        1.0000000
Lag 1000     0.9531985
Lag 2000     0.9188313
Lag 3000     0.8924896
Lag 4000     0.8720563
Lag 5000     0.8551858

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
           3.107            2.313            2.891            2.983
   nodematch.Sex
           3.342

Individual P-values (lower = worse):
           edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
```

```
    0.0018910749      0.0207264854      0.0038353298      0.0028555378
   nodematch.Sex
    0.0008320123
Joint P-value (lower = worse):  0.2294871 .

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






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

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

MORAL: Degeneracy is an indicator of a poorly specified model. It is not a property of all ERGMs, but it is associated with some dyadic-dependent terms, in particular, the reduced homogenous Markov specifications (e.g., 2-stars and triangle terms). For a good technical discussion of unstable terms see Schweinberger 2012. For a discussion of alternative terms that exhibit more stable behavior see Snijders et al. 2006. and for the gwesp term (and the curved exponential family terms in general) see Hunter and Handcock 2006.

7. Working with egocentrically sampled network data

One of the most powerful features of ERGMs is that they can be used to simulate complete networks from egocentrically sampled 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

Degree

Frequency

Fraction

Ties

0

180

0.36

0

1

245

0.49

245

2

60

0.12

120

3

15

0.03

45

Total

500

1.00

410

Sex mixing matrix (410 total)

Friend

M

F

Respondent

M

164

44

F

26

176

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**: In this case the tie ("seen more than once") it is undirected, so we will fit and simulate an undirected network. Note that ego data are in one sense inherently directed (respondents nominate alters, alters are not observed), but the type of tie that respondents report may be either directed or undirected, and that is what we are simulating.
- **Bipartite**: "Seen" is not a two-mode relationship, so we will fit and simulated a one-mode network. Note again that the ego data can be bipartite by design (egos vs. alters, if no alters are also respondents, or if data are collected on 2-mode networks) or not (if respondents are also alters). But again, the type of tie determines what we want to simulate.

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 and approaches are possible.

Initializing a network

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 frequencies should sum to 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
```

Target statistics for the model

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
```

ERGM model

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)
```

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

Take a look at the fitted model:

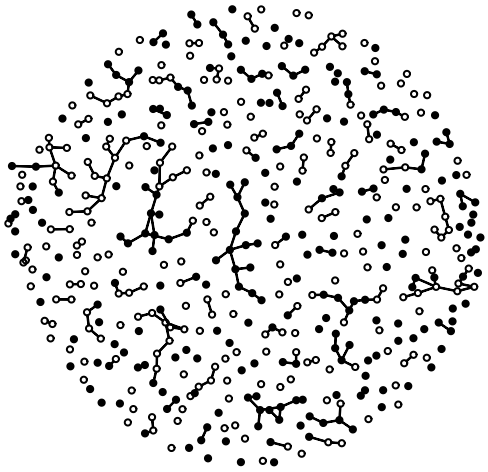
```
summary(ego.fit)
```

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

Formula:   nw ~ edges + nodematch("sex")
<environment: 0x000000001ce34e30>

Iterations:  8 out of 20

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.)
```

Simulation of a complete network from the model

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")
```



Diagnostics and model re-specification

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

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

```
    degree0 degree1 degree2 degree3
sim     234     170      76      16
obs     180     245      60      15
```

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

```
Note:  Marginal totals can be misleading
 for undirected mixing matrices.
   0  1
0 79 34
1 34 80
```

```
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: ~.
Parameters:
        edges nodematch.sex
    -7.487013      1.586633
```

More information can be obtained with

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

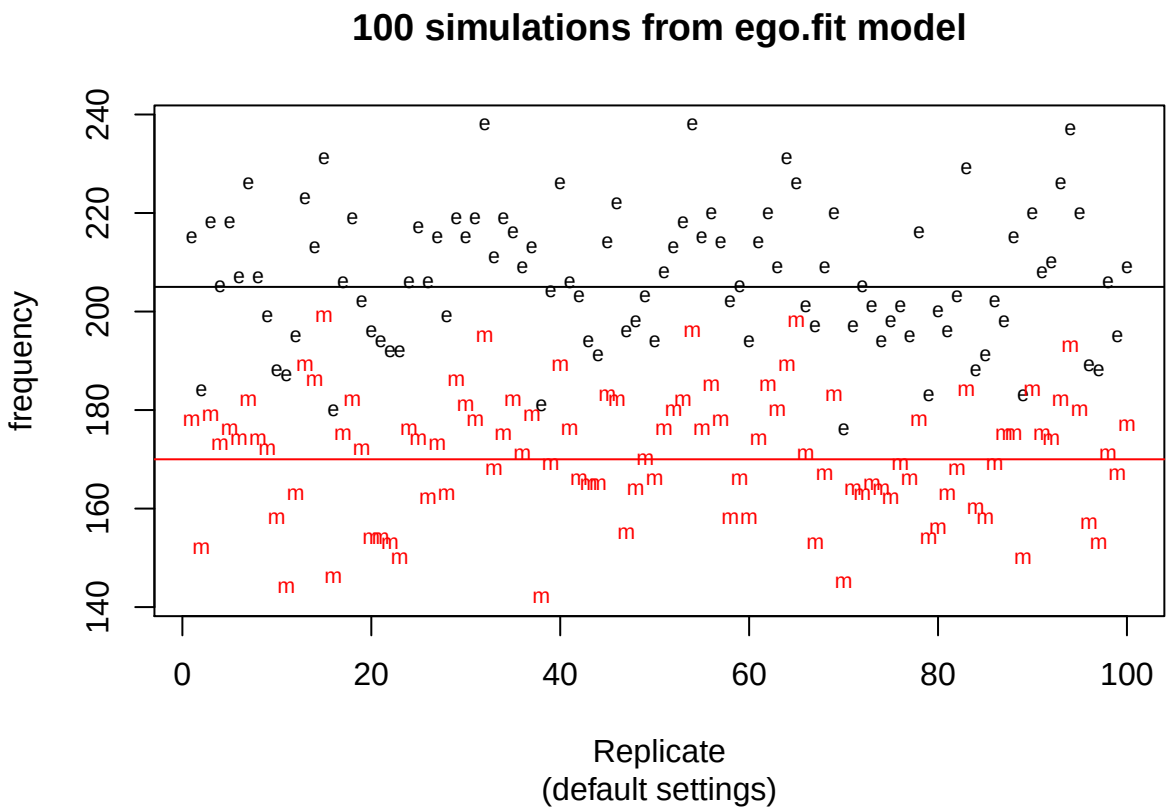
First, we'll look at how well the simulations reproduced the target statistics that were included in the model (note, not the observed full distributions):

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

```
     edges nodematch.sex
sim 206.64        170.96
obs 205.00        170.00
```

These look pretty good – the means of the simulated target stats are within a few percent of the observed, depending on your computer configuation. 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))
```



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.

The variation (about +/- 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,]  212  181   76   27    4    0    0    0    0    0     0
[2,]  233  195   51   14    6    1    0    0    0    0     0
[3,]  208  181   82   25    4    0    0    0    0    0     0
[4,]  216  184   77   20    3    0    0    0    0    0     0
[5,]  217  166   87   25    4    1    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

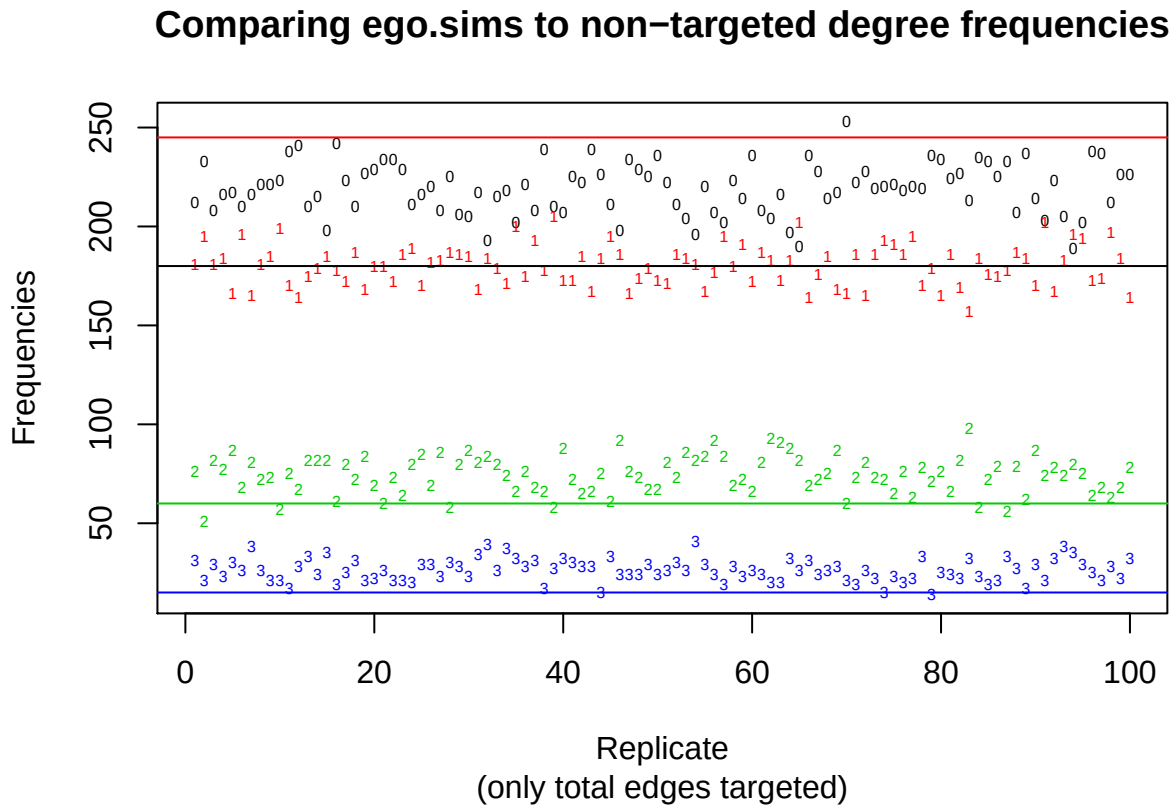
from this model match the frequencies we did observe? Aggregating the degrees of 3 or more in the simulations, we find:

```
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(sim=colMeans(sim.deg), obs=ego.deg)
```

```
      deg0   deg1  deg2 degree3+
sim 219.17 180.26 74.51    26.06
obs 180.00 245.00 60.00    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))
```



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 likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 0.01735
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 0.0008226
Step length converged twice. Stopping.
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() f
```

```
summary(ego.fit)
```

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

Formula:   nw ~ edges + nodematch("sex") + degree(0)
<environment: 0x000000002450efc0>

Iterations:  2 out of 20

Monte Carlo MLE Results:
             Estimate Std. Error MCMC % p-value
edges         -8.4060     0.2436      0  <1e-04 ***
nodematch.sex  1.5889     0.1838      0  <1e-04 ***
degree0       -0.9612     0.1611      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(sim=colMeans(sim.stats), obs=ego.target.stats)
```

```
     edges nodematch.sex degree0
sim 202.76        169.27  182.96
obs 205.00        170.00  180.00
```

And the full degree frequencies look much better:

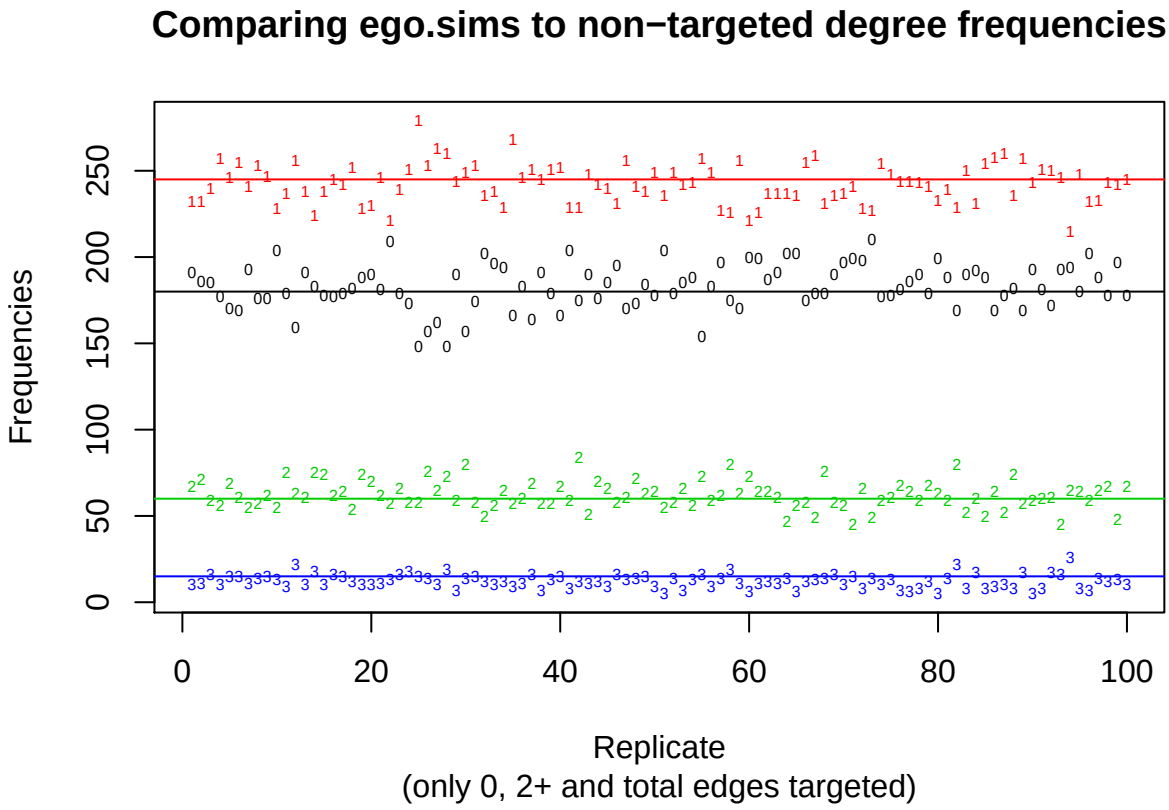
```
sim.fulldeg <- summary(ego.sim100 ~ degree(c(0:10)))
sim.deg <- cbind(sim.fulldeg[,1:3], apply(sim.fulldeg[,4:11],1,sum))
colnames(sim.deg) <- c(colnames(sim.fulldeg)[1:3],"degree3+")
rbind(sim=colMeans(sim.deg), obs=ego.deg)
```

```
    degree0 degree1 degree2 degree3+
sim  182.96   242.6   62.26    12.18
```

```
obs  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))
```



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 http://statnet.org/movies

8. Additional functionality in the statnet family of packages

Additional functionality

Packages that developed by statnet team that are not covered in this tutorial:

- classical social network analysis (`sna` package)
- temporal ergms for dynamic networks (`tergm` package)
- relational event models for networks (`relevent` package)
- latent space and latent cluster analysis (`latentnet` package)
- MLE estimation for degree distributions (negative binomial, Poisson, scale-free, etc.) (`degreenet` package)
- simulation of bipartite networks with given degree distributions (`networksis` package)
- hierarchical ERGMs (`hergm` package)
- ERGMs for valued ties (`ergm` package)
- network movie maker (`ndtv` package)
- network modeling of infectious disease and social diffusion processes (`EpiModel` package)

Any of these not in the ergm base package are in stand-alone packages that can be downloaded from CRAN. For more detailed information, please visit the `statnet` webpage www.statnet.org.

We also place our tutorials from many different workshops online on the statnet wiki.

Statnet Commons: The development group

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. link

Handcock MS (2003a). "Assessing Degeneracy in Statistical Models of Social Networks." Working Paper 39, Center for Statistics and the Social Sciences, University of Washington. link

Handcock MS (2003b). "Statistical Models for Social Networks: Inference and Degeneracy." In R Breiger, K Carley, P Pattison (eds.), *Dynamic Social Network Modeling and Analysis*, volume 126, pp. 229-252. Committee on Human Factors, Board on Behavioral, Cognitive, and Sensory Sciences, National Academy Press, Washington, DC.

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) link.

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). link

Krivitsky, P.N., Handcock, M.S,(2014). A separable model for dynamic networks *JRSS Series B-Statistical Methodology*, 76(1):29-46; 10.1111/rssb.12014 JAN 2014 link

Krivitsky, P. N., M. S. Handcock and M. Morris (2011). Adjusting for Network Size and Composition Effects in Exponential-family Random Graph Models, *Statistical Methodology* 8(4): 319-339, ISSN 1572-3127 link

Schweinberger, Michael (2011) Instability, Sensitivity, and Degeneracy of Discrete Exponential Families *JASA* 106(496): 1361-1370. link

Snijders, TAB et al (2006) New Specifications For Exponential Random Graph Models *Sociological Methodology* 36(1): 99-153 link