Introduction to Exponential-family Random Graph Models with `ergm`

The Statnet Development Team

`ergm` version 4.0.1 (2021-06-20)

Contents

0.1 Introduction

This vignette provides an introduction to statistical modeling of network data with *Exponential family Random Graph Models* (ERGMs) using `ergm` package. It is based on the `ergm` tutorial used in the `statnet` workshops, but covers a subset of that material.
The complete tutorial can be found on the `statnet` workshop wiki.

A more complete overview of the advanced functionality available in the `ergm` package can be found in this preprint.

0.1.1 Software installation

If you are reading this, you have probably already installed the `ergm` package. But in case you need to do that:

```
install.packages('ergm')
```

```
library(ergm)
```

Set a seed for simulations – this is not necessary, but it ensures that you will get the same results as this vignette (if you execute the same commands in the same order).

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

0.2 1. Statistical network modeling with ERGMs

This is a *very brief* overview of the modeling framework, as the primary purpose of this tutorial is to show how to implement statistical analysis of network data with ERGMs using the `ergm` package. For more detail (and to really understand ERGMs) please see the references at the end of this tutorial.

Exponential-family random graph models (ERGMs) are a general class of models based on exponential-family theory for specifying the tie probability distribution for a set of random graphs or networks. Within this framework, one can—among other tasks:

- Define a model for a network that includes covariates representing features like nodal attribute homophily, mutuality, triad effects, and a wide range of other structural features of interest;
- Obtain maximum-likehood estimates for the parameters of the specified model for a given data set;
- Test the statistical significance of individual coefficients, assess models for convergence and goodness-of-fit and perform various types of model comparison; and
- Simulate new networks from the underlying probability distribution implied by the fitted model.

0.2.1 The general form for an ERGM

ERGMs are a class of models, like linear regression or GLMs. The general form of the model specifies the probability of the entire network (on the left hand side), as a function of terms that represent network features we hypothesize may occur more or less likely than expected by chance (on the right hand side). The general form of the model 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 ("ERGM terms") 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).

If you're not familiar with the compact notation here, note that the numerator represents a formula that is linear in the log form:

$$\log(\exp(\theta' g(y))) = \theta_1 g_1(y) + \theta_2 g_2(y) + ... + \theta_p g_p(y)$$

where p is the number of terms in the model. From this one can more easily observe the analogy to a traditional statistical model: the coefficients θ represent the size and direction of the effects of the covariates $g(y)$ on the overall probability of the network.

0.2.1.1 The model statistics $g(y)$: ERGM terms

The statistics $g(y)$ can be thought of as the "covariates" in the model. In the network modeling context, these represent network features like density, homophily, triads, etc. In one sense, they are like covariates you might use in other statistical models. But they are different in one important respect: these $g(y)$ statistics are functions of the network itself – each is defined by the frequency of a specific configuration of dyads observed in the network – so they are not measured by a question you include in a survey (e.g., the income of a node), but instead need to be computed on the specific network you have, after you have collected the data.

As a result, every term in an ERGM must have an associated algorithm for computing its value for your network. The `ergm` package in `statnet` includes about 150 term-computing algorithms. We will explore some of these terms in this tutorial, and links to more information are provided in section 3..

You can get the list of all available terms, and the syntax for using them, by typing:

```
?'ergm-terms'
```

You can also search for terms with keywords:

```
search.ergmTerms(keyword='homophily')
```

```
## Found  13  matching ergm terms:
## b1degrange(from, to=+Inf, by=NULL, homophily=FALSE, levels=NULL)
##  Degree range for the first mode in a bipartite (a.k.a. two-mode) network
##
## b1nodematch(attr, diff=FALSE, keep=NULL, alpha=1, beta=1,)
##  Nodal attribute-based homophily effect for the first mode in a bipartite (aka two-mode) network
##
## b2degrange(from, to=+Inf, by=NULL, homophily=FALSE, levels=NULL)
##  Degree range for the second mode in a bipartite (a.k.a. two-mode) network
##
## b2nodematch(attr, diff=FALSE, keep=NULL, alpha=1, beta=1,)
##  Nodal attribute-based homophily effect for the second mode in a bipartite (aka two-mode) network
##
## degrange(from, to=+Inf, by=NULL, homophily=FALSE, levels=NULL)
##  Degree range
##
## degree(d, by=NULL, homophily=FALSE, levels=NULL)
##  Degree
##
## idegrange(from, to=+Inf, by=NULL, homophily=FALSE, levels=NULL)
##  In-degree range
##
## idegree(d, by=NULL, homophily=FALSE, levels=NULL)
##  In-degree
##
## nodematch(attr, diff=FALSE, keep=NULL, levels=NULL)
##  Uniform homophily and differential homophily
##
## nodematch(attr, diff=FALSE, keep=NULL, levels=NULL, form="sum")
##  Uniform homophily and differential homophily
##
## match()
##  Uniform homophily and differential homophily
##
## odegrange(from, to=+Inf, by=NULL, homophily=FALSE, levels=NULL)
##  Out-degree range
##
## odegree(d, by=NULL, homophily=FALSE, levels=NULL)
##  Out-degree
```

For more information, see the vignette on `ergm-terms`:

```
vignette('ergm-term-crossRef')
```

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 these terms. They introduce complex cascading effects that can often lead to counter-intuitive

and highly non-linear outcomes. In addition, a model with dyad dependent terms requires a different estimation algorithm, so when we use them below you will see some different components in the output.

An overview and discussion of many of these terms can be found in the 'Specifications' paper in the *Journal of Statistical Software v24(4)*

0.2.1.2 ERGM probabilities: at the tie-level The ERGM expression for the probability of the entire graph shown above can be re-expressed in terms of the conditional log-odds of a single tie between two actors:

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

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}.
- $\delta(y_{ij})$ is a vector of the "change statistics" for each model term. The change statistic records how the $g(y)$ 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.

So $\delta(y_{ij})$ 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 y.

This expression shows that the coefficient θ can be interpreted as that term's contribution to the log-odds of an individual tie, conditional on all other dyads remaining the same. The coefficient for each term in the model is multiplied by the number of configurations that tie will create (or remove) for that specific term.

0.2.1.3 The `summary` and `ergm` functions, and supporting functions We'll start by running some simple models to demonstrate the most commonly used functions for ERG modeling.

The syntax for specifying a model in the `ergm` package follows **R**'s formula convention:

$$my.network \sim my.vector.of.model.terms$$

This syntax is used for both the `summary` and `ergm` functions. The `summary` function simply returns the numerical values of the network statistics in the model. The `ergm` function estimates the model with those statistics.

It is good practice to always run a `summmary` command on a model before fitting it with `ergm`. This is the ERGM equivalent of performing some descriptive analysis on your covariates. This can help you make sure you understand what the term represents, and it can help to flag potential problems that will lead to poor modeling results.

0.2.2 Network data

Network data can come in many different forms – as adjacency matrices, edgelists and data frames. For use with `ergm` these need to be converted into a `network` class object. The `network` package provides the utilities for this, and more information can be found in the examples and vignette there.

```
vignette("networkVignette")
```

Here, we will use some of the network objects included with the `ergm` package.

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

0.2.3 Model specification

We'll start with Padgett's data on Renaissance Florentine families for our first example. 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 # Look at the flomarriage network properties (uses `network`), esp. the vertex attributes
```

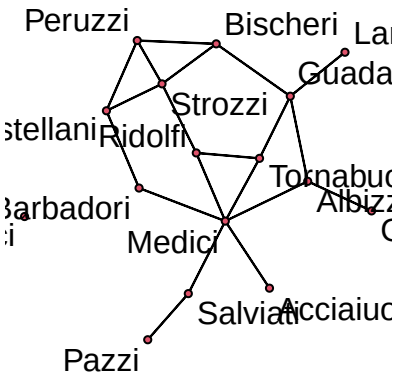
```
##  Network attributes:
##   vertices = 16
##   directed = FALSE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 20
##     missing edges= 0
##     non-missing edges= 20
##
##  Vertex attribute names:
##     priorates totalties vertex.names wealth
##
## No edge attributes
```

```
par(mfrow=c(1,2)) # Setup a 2 panel plot
plot(flomarriage,
     main="Florentine Marriage",
     cex.main=0.8,
     label = network.vertex.names(flomarriage)) # Plot the network
wealth <- flomarriage %v% 'wealth' # %v% references vertex attributes
wealth
```

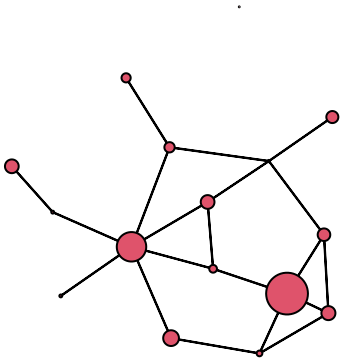
```
##  [1]  10  36  55  44  20  32   8  42 103  48  49   3  27  10 146  48
```

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

Florentine Marriage



Florentine marriage by wealth



0.2.3.1 A simple Bernoulli ("Erdos/Renyi") model We begin with a simple model, containing only one term that represents the total number of edges in the network, $\sum y_{ij}$. The name of this ergm-term is `edges`, and when included in an ERGM its coefficient controls the overall density of the network.

```
summary(flomarriage ~ edges) # Look at the $g(y)$ statistic for this model
```

```
## edges
##    20
```

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

```
summary(flomodel.01) # Look at the fitted model object
```

```
## Call:
## ergm(formula = flomarriage ~ edges)
##
## Maximum Likelihood 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. MC Std. Err. = 0)
```

This simple model specifies a single homogeneous probability for all ties, which is captured by the coefficient of the `edges` term. How should we interpret this coefficient? The easiest way is to return to the logit form of the ERGM. The log-odds that a tie is present is

$$\begin{aligned}\text{logit}(p(y)) &= \theta \times \delta(g(y)) \\ &= -1.61 \times \text{change in the number of ties} \\ &= -1.61 \times 1\end{aligned}$$

for every tie, since the addition of any tie to the network always increases the total number of ties by 1.

The corresponding probability is obtained by taking the expit, or inverse logit, of θ:

$$\begin{aligned}&= \exp(-1.61)/(1+\exp(-1.61)) \\ &= 0.17\end{aligned}$$

This probability corresponds to the density we observe in the flomarriage network: there are 20 ties and $\binom{16}{2} = (16 \times 15)/2 = 120$ dyads, so the probability of a tie is $20/120 = 0.17$.

0.2.3.2 Triad formation Let's add a term often thought to be a measure of "clustering": the number of completed triangles in the network, or $\sum y_{ij}y_{ik}y_{jk} \div 3$.
The name for this ergm-term is `triangle`.

This is an example of a dyad dependent term: The status of any triangle containing dyad y_{ij} depends on the status of dyads of the form y_{ik} and y_{jk}. This means that any model containing the ergm-term `triangle` has the property that dyads are not probabilistically independent of one another. 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)
summary(flomodel.02)
```

```
## Call:
## ergm(formula = flomarriage ~ edges + triangle)
##
## Monte Carlo Maximum Likelihood Results:
##
##          Estimate Std. Error MCMC % z value Pr(>|z|)
## edges     -1.7354     0.3503      0  -4.954   <1e-04 ***
## triangle   0.1893     0.4593      0   0.412     0.68
## ---
## 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. MC Std. Err. = 0.01451)
```

Now, how should we interpret coefficients?

The conditional log-odds of two actors having a tie, keeping the rest of the network fixed, is

$$-1.74 \times \text{change in the number of ties} + 0.19 \times \text{change in number of triangles.}$$

- For a tie that will create no triangles, the conditional log-odds is: -1.74.
- if one triangle: $-1.74 + 0.19 = -1.55$
- if two triangles: $-1.74 + 2 \times 0.19 = -1.36$
- the corresponding probabilities are 0.16, 0.18, and 0.20.

Let's take a closer look at the ergm object that the function outputs:

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

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

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

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

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

```
##     edges  triangle
## -1.735361  0.189265
```

0.2.3.3 Nodal covariates: effects on mean degree We saw earlier that wealth appeared to be associated with higher degree in this network. We can use `ergm` to test this. Wealth is a nodal covariate, so we use the ergm-term **nodecov**.

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

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

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

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

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

```
## Maximizing the pseudolikelihood.

## Finished MPLE.

## Stopping at the initial estimate.

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

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

```
## Call:
## ergm(formula = flomarriage ~ edges + nodecov("wealth"))
##
## Maximum Likelihood 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. MC Std. Err. = 0)
```

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

Question: What does the value of the **nodecov** statistic represent?

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:

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

or

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

- 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.

This model specification does not include a term for homophily by wealth, i.e., a term accounting for similarity in wealth of the two end nodes of a potential tie. It just specifies a relation between wealth and mean degree. To specify homophily on wealth, you could use the ergm-term ***absdiff***. See section 3 below for more information on ergm-terms.?faux

0.2.3.4 Nodal covariates: Homophily

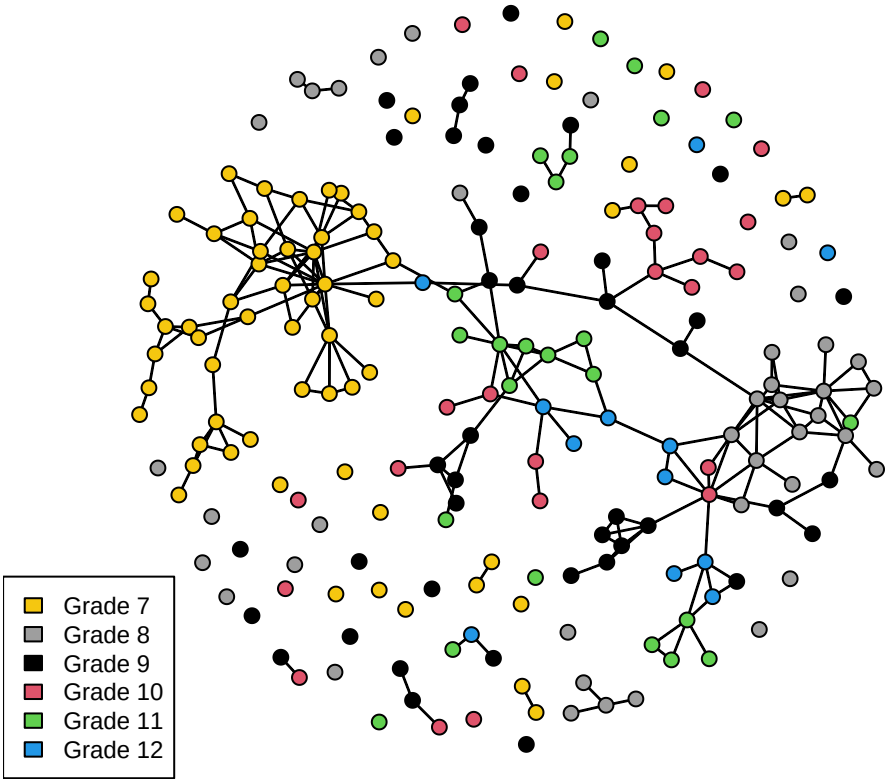
Let's try a larger network, a simulated mutual friendship network based on one of the schools from the AddHealth study. Here, we'll examine the homophily in friendships by grade and race. Both are discrete attributes so we use the ergm-term ***nodematch***.

```
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 +
                       nodefactor('Grade') + nodematch('Grade',diff=T) +
                       nodefactor('Race') + nodematch('Race',diff=T))
```

```
## Observed statistic(s) nodematch.Race.Black and nodematch.Race.Other are at their smallest attainable
## 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(fauxmodel.01)
```

```
## Call:
## ergm(formula = mesa ~ edges + nodefactor("Grade") + nodematch("Grade",
##     diff = T) + nodefactor("Race") + nodematch("Race", diff = T))
##
## Maximum Likelihood Results:
##
##                       Estimate Std. Error MCMC % z value Pr(>|z|)
## edges                  -8.0538     1.2561      0  -6.412  < 1e-04 ***
## nodefactor.Grade.8      1.5201     0.6858      0   2.216 0.026663 *
## nodefactor.Grade.9      2.5284     0.6493      0   3.894  < 1e-04 ***
## nodefactor.Grade.10     2.8652     0.6512      0   4.400  < 1e-04 ***
## nodefactor.Grade.11     2.6291     0.6563      0   4.006  < 1e-04 ***
## nodefactor.Grade.12     3.4629     0.6566      0   5.274  < 1e-04 ***
## nodematch.Grade.7       7.4662     1.1730      0   6.365  < 1e-04 ***
## nodematch.Grade.8       4.2882     0.7150      0   5.997  < 1e-04 ***
## nodematch.Grade.9       2.0371     0.5538      0   3.678 0.000235 ***
## nodematch.Grade.10      1.2489     0.6233      0   2.004 0.045111 *
## nodematch.Grade.11      2.4521     0.6124      0   4.004  < 1e-04 ***
## nodematch.Grade.12      1.2987     0.6981      0   1.860 0.062824 .
## nodefactor.Race.Hisp   -1.6659     0.2963      0  -5.622  < 1e-04 ***
## nodefactor.Race.NatAm  -1.4725     0.2869      0  -5.132  < 1e-04 ***
## nodefactor.Race.Other  -2.9618     1.0372      0  -2.856 0.004296 **
## nodefactor.Race.White  -0.8488     0.2958      0  -2.869 0.004112 **
## nodematch.Race.Black      -Inf     0.0000      0    -Inf  < 1e-04 ***
## nodematch.Race.Hisp     0.6912     0.3451      0   2.003 0.045153 *
## nodematch.Race.NatAm    1.2482     0.3550      0   3.517 0.000437 ***
## nodematch.Race.Other      -Inf     0.0000      0    -Inf  < 1e-04 ***
## nodematch.Race.White    0.3140     0.6405      0   0.490 0.623947
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 28987  on 20910  degrees of freedom
##  Residual Deviance:  1827  on 20889  degrees of freedom
##
## AIC: 1865  BIC: 2016  (Smaller is better. MC Std. Err. = 0)
##
##  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")
```

```
##       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
```

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

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  +
          nodefactor('Grade') + nodematch('Grade',diff=T) +
          nodefactor('Race') + nodematch('Race',diff=T))
```

```
##                 edges    nodefactor.Grade.8    nodefactor.Grade.9
##                   203                    75                    65
##   nodefactor.Grade.10   nodefactor.Grade.11   nodefactor.Grade.12
##                    36                    49                    28
##     nodematch.Grade.7     nodematch.Grade.8     nodematch.Grade.9
##                    75                    33                    23
##    nodematch.Grade.10    nodematch.Grade.11    nodematch.Grade.12
##                     9                    17                     6
##  nodefactor.Race.Hisp nodefactor.Race.NatAm nodefactor.Race.Other
##                   178                   156                     1
## nodefactor.Race.White  nodematch.Race.Black   nodematch.Race.Hisp
##                    45                     0                    53
##  nodematch.Race.NatAm  nodematch.Race.Other  nodematch.Race.White
##                    46                     0                     4
```

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

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

0.2.3.5 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
```

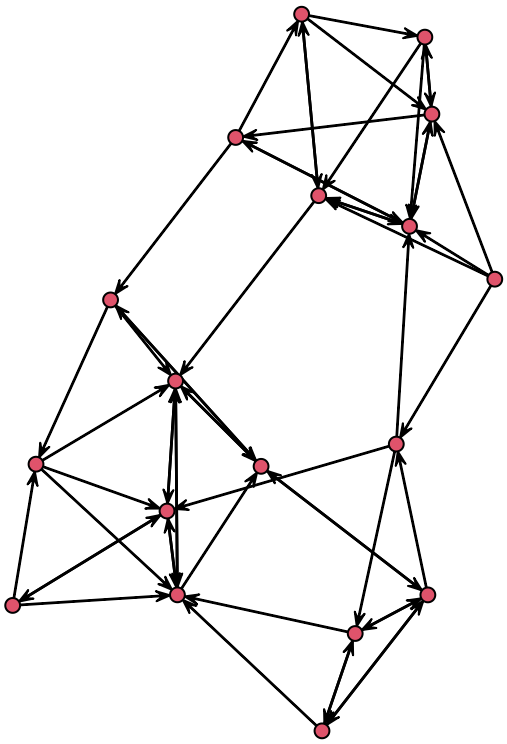
```
##  [1] "faux.mesa.high" "fauxmodel.01"   "flobusiness"    "flomarriage"
##  [5] "flomodel.01"    "flomodel.02"    "flomodel.03"    "items"
##  [9] "mesa"           "samplk1"        "samplk2"        "samplk3"
## [13] "termBlock"      "terms"          "wealth"
```

```
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. This means 30 of the 56 ties are reciprocated, i.e., they are part of dyads in which both directional ties are present.

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

```
## Call:
## ergm(formula = samplk3 ~ edges + mutual)
##
## Monte Carlo Maximum Likelihood Results:
##
```

```
##        Estimate Std. Error MCMC % z value Pr(>|z|)
## edges   -2.1415     0.2344      0  -9.135   <1e-04 ***
## mutual   2.2951     0.5066      0   4.531   <1e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 424.2  on 306  degrees of freedom
##  Residual Deviance: 267.8  on 304  degrees of freedom
##
## AIC: 271.8  BIC: 279.2  (Smaller is better. MC Std. Err. = 0.3303)
```

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 log-odds of a mutual tie are about zero, which means the probability is about 50%. (Do you see why a log-odds of zero corresponds to a probability of 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 as ergm-terms (and we'll talk about these more below).

0.3 2. Missing data

It is important to distinguish between the absence of a tie and the absence of data on whether a tie exists. The former is an observed zero, whereas the latter is unobserved. 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.

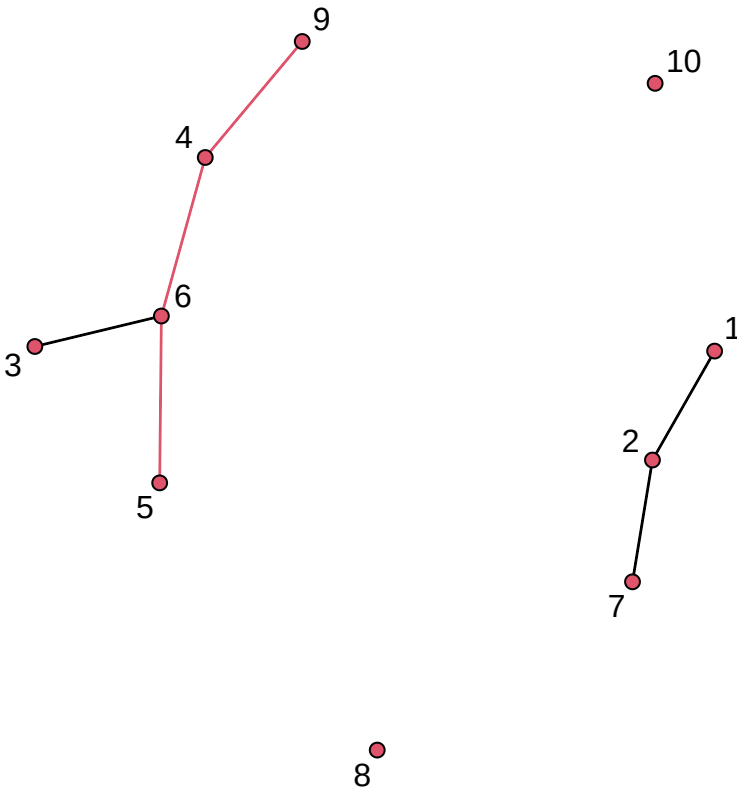
Start by estimating an ergm on a network with two missing ties, where both ties are identified as missing.

```
missnet <- network.initialize(10,directed=F) # initialize an empty net with 10 nodes
missnet[1,2] <- missnet[2,7] <- missnet[3,6] <- 1 # add a few ties
missnet[4,6] <- missnet[4,9] <- missnet[5,6] <- NA # mark a few dyads missing
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
```

```
# plot missnet with missing dyads 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)
```




```
# fit an ergm to the network with missing data identified
summary(missnet~edges)
```

```
## edges
##     3
```

```
summary(ergm(missnet~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.

## Call:
## ergm(formula = missnet ~ edges)
##
## Maximum Likelihood Results:
##
##       Estimate Std. Error MCMC % z value Pr(>|z|)
## edges  -2.5649     0.5991      0  -4.281   <1e-04 ***
## ---
## 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. MC Std. Err. = 0)
```

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 (observed) value "0" and check how the value of the coefficient will change. Can you predict whether it will get bigger or smaller? Can you calculate it directly before checking the output of an `ergm` fit? Let's see what happens.

```
missnet_bad <- missnet # create network with missing dyads set to 0
missnet_bad[4,6] <- missnet_bad[4,9] <- missnet_bad[5,6] <- 0

# fit an ergm to the network with missing dyads set to 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))
```

```
## 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.

## Call:
## ergm(formula = missnet_bad ~ edges)
##
## Maximum Likelihood Results:
##
##       Estimate Std. Error MCMC % z value Pr(>|z|)
## edges  -2.6391     0.5976      0  -4.416   <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. MC Std. Err. = 0)
```

The coefficient is smaller now because the missing ties are counted as "0", and this 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.

0.4 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)

Because these terms are not exogeneous measures, but functions of the dyad states in the network, they must be calculated for the network that is being modeled. 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.

0.4.1 Coding new ergm-terms

There is a `statnet` package — `ergm.userterms` — that facilitates the writing of new ergm-terms. The package is available on CRAN, and installing it will include the tutorial (ergmuserterms.pdf). The tutorial can also be found in the *Journal of Statistical Software 52(2)*, and some introductory slides and installation instructions 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. While 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.

0.5 4. Assessing convergence for dyad dependent models: MCMC Diagnostics

When dyad dependent terms are in the model, the computational algorithms in `ergm` use MCMC (with a Metropolis-Hastings sampler) to estimate the parameters.

For these models, it is important to assess model convergence before interpreting the model results – before evaluating statistical significance, interpreting coefficients, or assessing goodness of fit.
To do this, we use the function `mcmc.diagnostics`.

Below we show a simple example of a model that converges, and how to use the MCMC diagnostics to identify this.

0.5.1 What it looks like when a model converges properly

We will first consider a simple dyadic dependent model where the algorithm works using the program defaults, with a degree(1) term that captures whether there are more (or less) degree 1 nodes than we would expect, given the density.

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

```
##   edges degree1
##      15       3
```

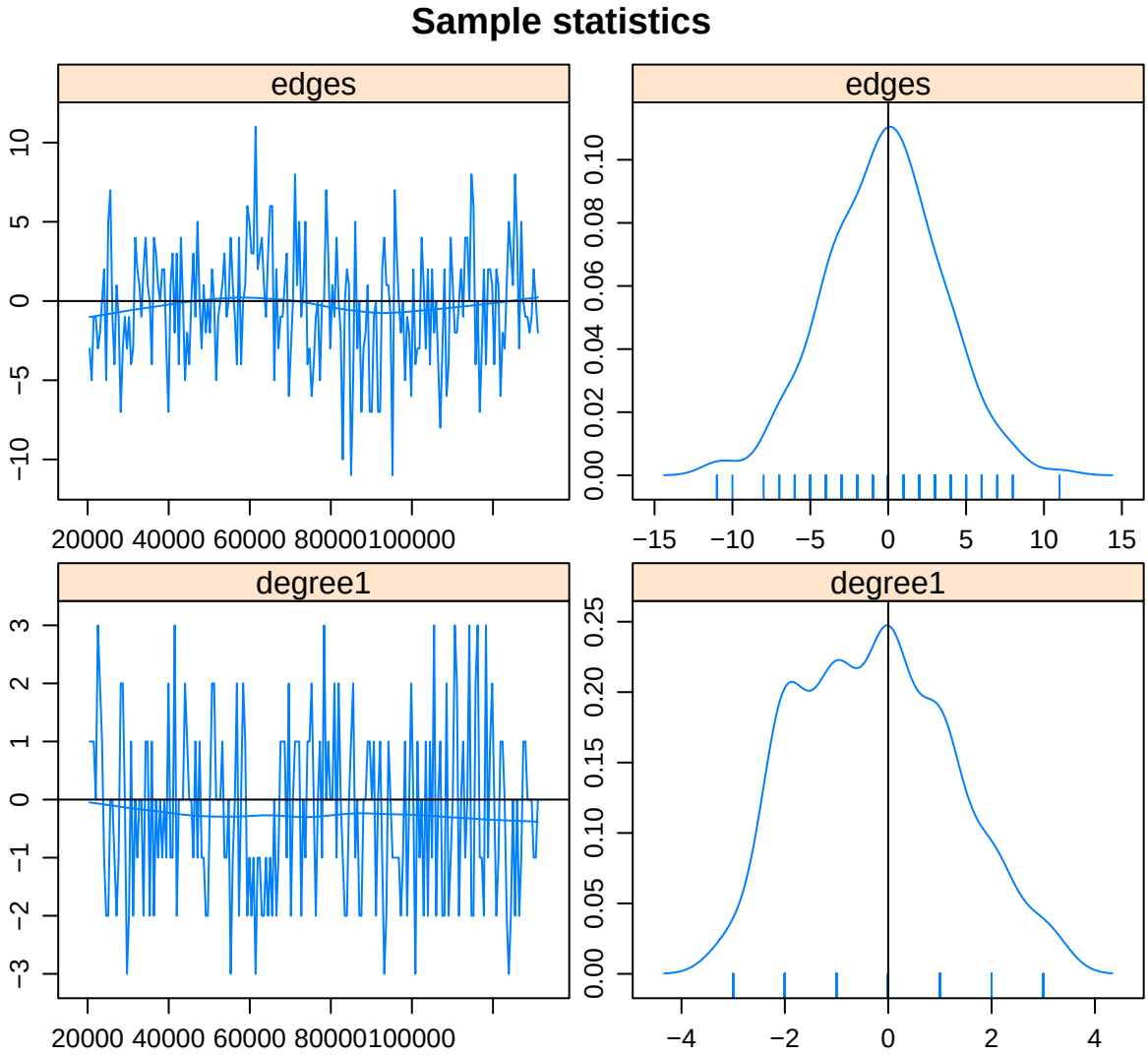
```
fit <- ergm(flobusiness~edges+degree(1))
summary(fit)
```

```
## Call:
## ergm(formula = flobusiness ~ edges + degree(1))
##
## Monte Carlo Maximum Likelihood Results:
##
##         Estimate Std. Error MCMC % z value Pr(>|z|)
## edges    -2.0866     0.2938      0  -7.101   <1e-04 ***
## degree1  -0.5960     0.7445      0  -0.801    0.423
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 166.36  on 120  degrees of freedom
##  Residual Deviance:  89.39  on 118  degrees of freedom
##
## AIC: 93.39  BIC: 98.97  (Smaller is better. MC Std. Err. = 0.03131)
```

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

```
## Sample statistics summary:
##
## Iterations = 20480:131072
## Thinning interval = 512
## Number of chains = 1
## Sample size per chain = 217
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean    SD Naive SE Time-series SE
## edges   -0.2995 3.681  0.24992         0.2908
## degree1 -0.2396 1.452  0.09858         0.1205
##
## 2. Quantiles for each variable:
##
##         2.5% 25% 50% 75% 97.5%
## edges   -7.0  -3   0   2     7
## degree1 -2.6  -1   0   1     3
##
##
## Are sample statistics significantly different from observed?
##                  edges    degree1 Overall (Chi^2)
## diff.      -0.2995392 -0.2396313              NA
## test stat. -1.0299123 -1.9881627      8.59774962
## P-val.      0.3030512  0.0467937      0.01545957
##
## Sample statistics cross-correlations:
##              edges    degree1
## edges    1.0000000 -0.3685222
## degree1 -0.3685222  1.0000000
##
## Sample statistics auto-correlation:
## Chain 1
##                edges     degree1
## Lag 0     1.00000000  1.00000000
## Lag 512   0.14822664  0.04663071
## Lag 1024 -0.01457000 -0.03252258
## Lag 1536  0.09904481 -0.03703903
## Lag 2048  0.14567434  0.07698536
## Lag 2560  0.09355121 -0.01033165
##
## Sample statistics burn-in diagnostic (Geweke):
## Chain 1
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##   edges degree1
## -0.6804  0.2431
##
## Individual P-values (lower = worse):
```

```
##     edges   degree1
## 0.4962808 0.8079541
## Joint P-value (lower = worse):  0.7044002 .
```




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

What this shows is statistics from the final iteration of the MCMC chain, on the left as a "traceplot" (the deviation of the statistic value in each sampled network from the observed value), and on the right as the distribution of the sample statistic deviations.

This is what you want to see in the MCMC diagnostics: a "fuzzy caterpillar". In the last section we'll look at some models that don't converge properly, and how to use MCMC diagnostics to identify and address this. There are many control parameters for the MCMC algorithm ("help(control.ergm)"), to improve model convergence.

0.6 5. 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.

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

class(flomodel.03.sim) # what does this produce?
```

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

```r
summary(flomodel.03.sim) # quick summary
```

```
## Number of Networks: 10
## Model: flomarriage ~ edges + nodecov("wealth")
## Reference: ~Bernoulli
## Constraints: ~.
## Stored network statistics:
##       edges nodecov.wealth
##  [1,]    22           2402
##  [2,]    17           2292
##  [3,]    17           1699
##  [4,]    26           2493
##  [5,]    17           1927
##  [6,]    16           1653
##  [7,]    15           1761
##  [8,]    16           1542
##  [9,]    20           2292
## [10,]    22           2594
## attr(,"monitored")
## [1] FALSE FALSE
```

```
## Number of Networks: 10
## Model: flomarriage ~ edges + nodecov("wealth")
## Reference: ~Bernoulli
## Constraints: ~.
```

```r
attributes(flomodel.03.sim) # what's in this object?
```

```
## $coefficients
##          edges nodecov.wealth
##    -2.59492903     0.01054591
##
## $control
## Control parameter list generated by 'control.simulate.formula' or equivalent. Non-empty parameters:
## MCMC.burnin: 16384
## MCMC.interval: 1024
## MCMC.scale: 1
## MCMC.prop: ~sparse
## MCMC.prop.weights: "default"
## MCMC.batch: 0
## MCMC.effectiveSize.damp: 10
## MCMC.effectiveSize.maxruns: 1000
## MCMC.effectiveSize.burnin.pval: 0.2
## MCMC.maxedges: Inf
## MCMC.runtime.traceplot: FALSE
## network.output: "network"
## parallel: 0
## parallel.version.check: TRUE
## parallel.inherit.MT: FALSE
## MCMC.samplesize: 10
```

```
## obs.MCMC.mul: 0.25
## obs.MCMC.samplesize.mul: 0.5
## obs.MCMC.interval.mul: 0.5
## obs.MCMC.burnin.mul: 0.5
## obs.MCMC.prop: ~sparse
## obs.MCMC.prop.weights: "default"
## MCMC.save_networks: TRUE
##
## $response
## [1] NA
##
## $class
## [1] "network.list"
##
## $stats
##       edges nodecov.wealth
##  [1,]    22           2402
##  [2,]    17           2292
##  [3,]    17           1699
##  [4,]    26           2493
##  [5,]    17           1927
##  [6,]    16           1653
##  [7,]    15           1761
##  [8,]    16           1542
##  [9,]    20           2292
## [10,]    22           2594
## attr(,"monitored")
## [1] FALSE FALSE
##
## $formula
## flomarriage ~ edges + nodecov("wealth")
## attr(,".Basis")
##  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
##
## $constraints
## ~.
## <environment: 0x55748444f788>
##
## $reference
## ~Bernoulli
```

```
## <environment: 0x557485397200>
```

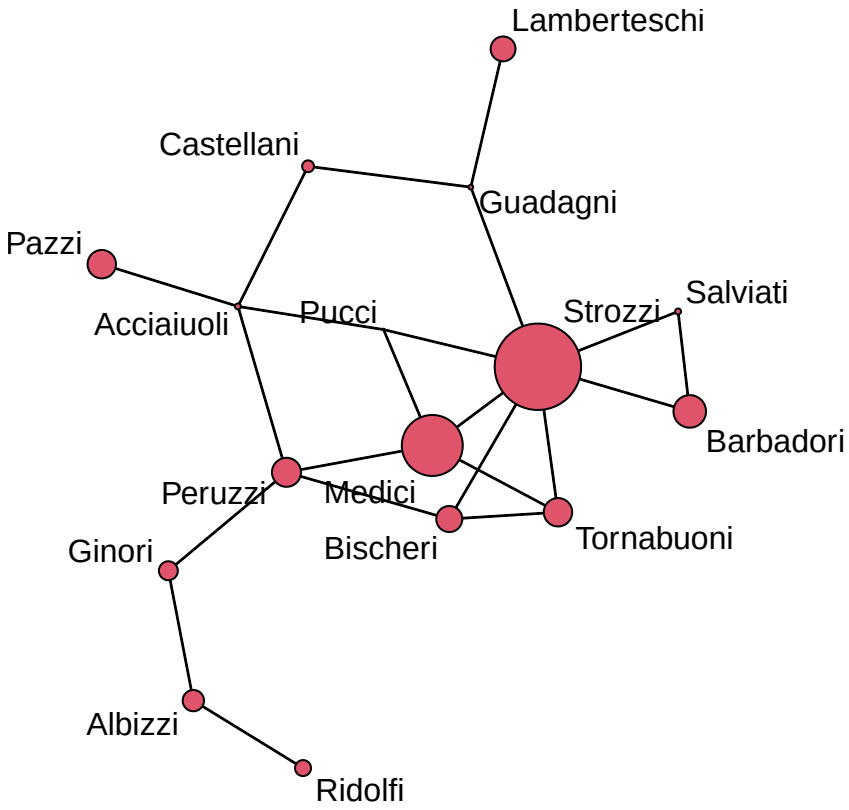
```
# are the simulated stats centered on the observed stats?
rbind("obs"=summary(flomarriage~edges+nodecov("wealth")),
      "sim mean"=colMeans(attr(flomodel.03.sim, "stats")))
```

```
##          edges nodecov.wealth
## obs       20.0         2168.0
## sim mean  18.8         2065.5
```

```
# we can also plot individual simulations
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",
     vertex.cex = (flomodel.03.sim[[1]] %v% "wealth")/25)
```



Voila. Of course, yours will look somewhat different.

0.7 6. 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.

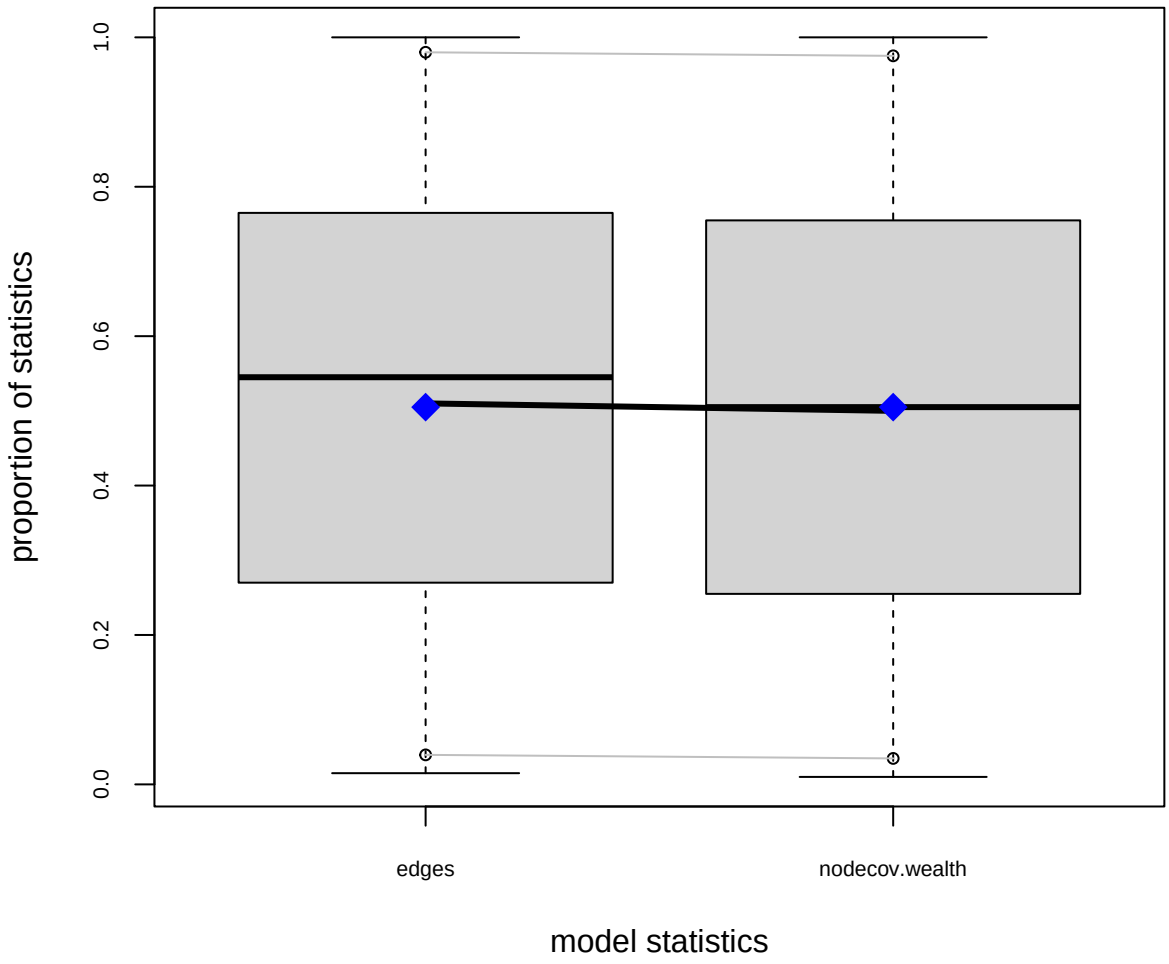
The **gof** function is a bit different than the **summary**, **ergm**, and **simulate** functions, in that it currently only takes 3 ergm-terms as arguments: degree, esp (edgwise share partners), and distance (geodesic distances). Each of these terms captures an aggregate network distribution, at either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

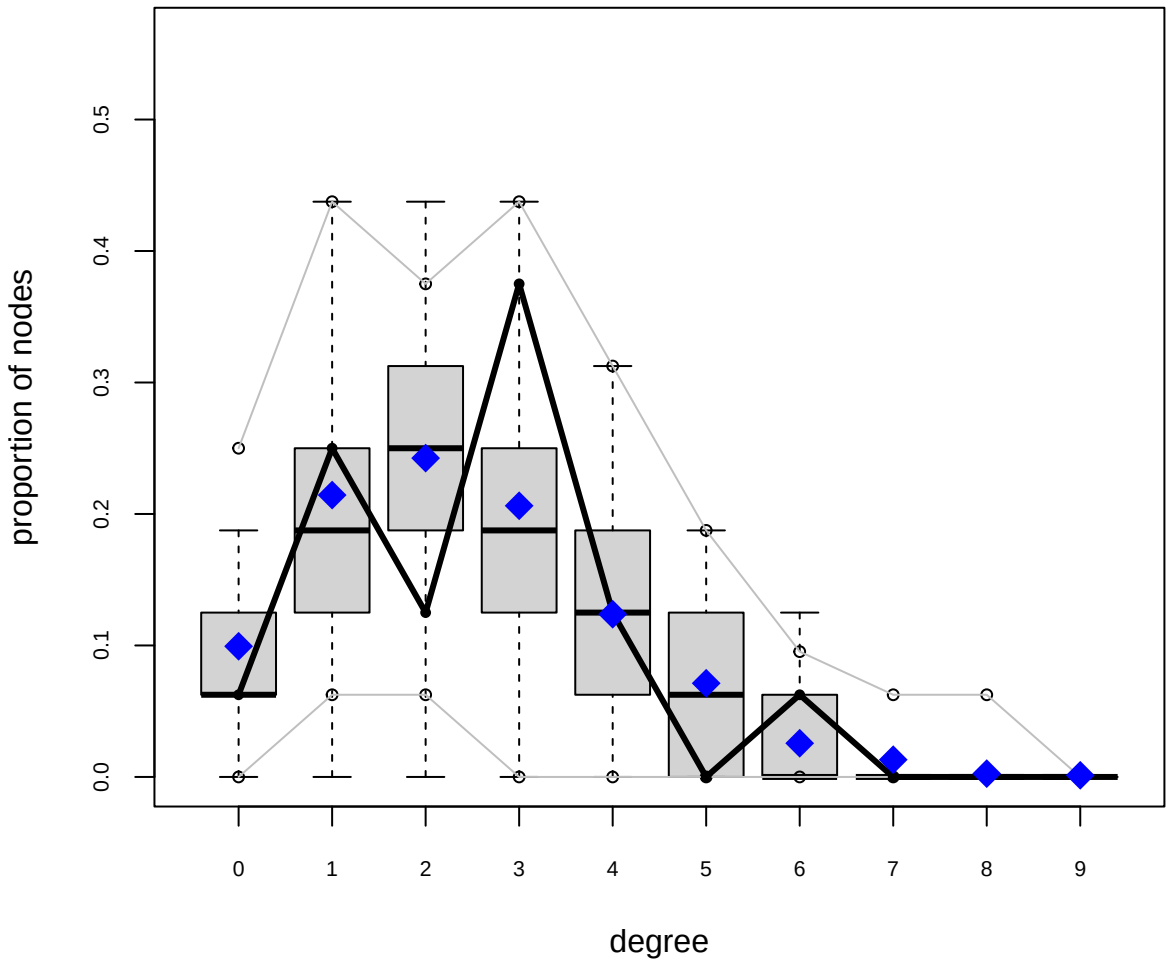
```
flomodel.03.gof <- gof(flomodel.03)
flomodel.03.gof
```

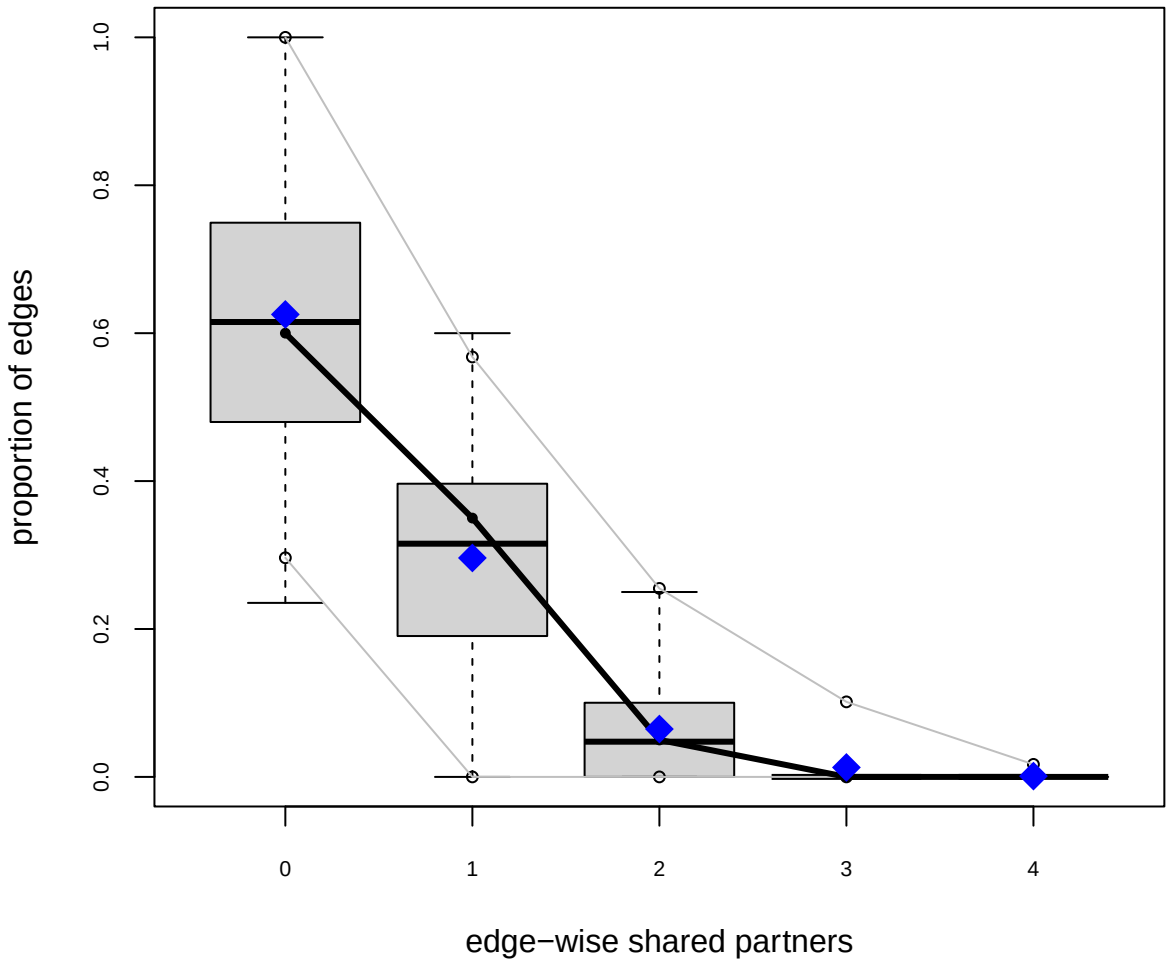
```
##
## Goodness-of-fit for degree
##
##         obs min mean max MC p-value
## degree0   1   0 1.59   5       1.00
## degree1   4   0 3.43   9       0.90
## degree2   2   0 3.88   7       0.32
```

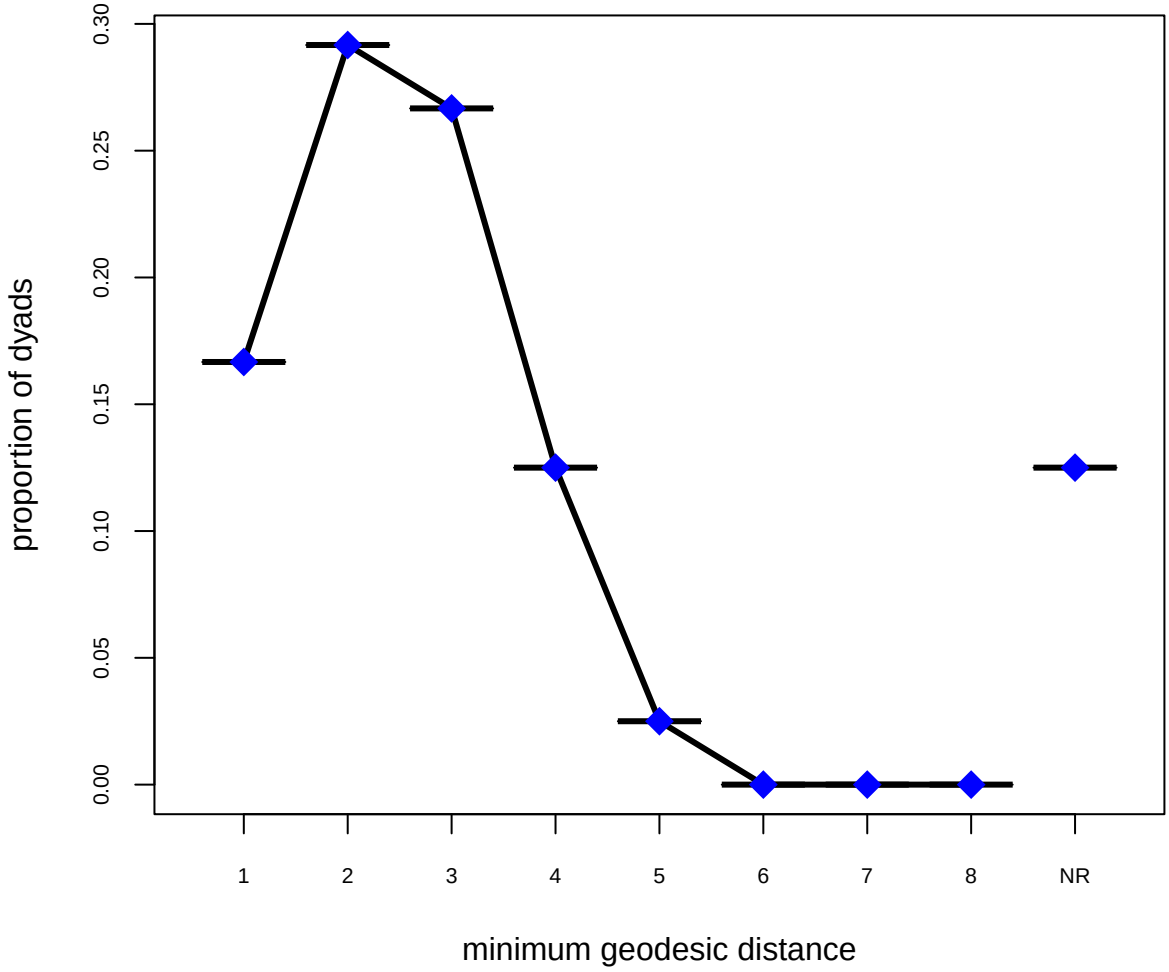
```
## degree3   6   0 3.30   7       0.20
## degree4   2   0 1.98   5       1.00
## degree5   0   0 1.14   3       0.58
## degree6   1   0 0.41   2       0.76
## degree7   0   0 0.21   2       1.00
## degree8   0   0 0.04   1       1.00
## degree9   0   0 0.02   1       1.00
##
## Goodness-of-fit for edgewise shared partner
##
##      obs min  mean max MC p-value
## esp0  12   4 11.82  20       1.00
## esp1   7   0  5.99  18       0.88
## esp2   1   0  1.43  12       1.00
## esp3   0   0  0.30   4       1.00
## esp4   0   0  0.03   1       1.00
##
## Goodness-of-fit for minimum geodesic distance
##
##     obs min mean max MC p-value
## 1    20  20   20  20          1
## 2    35  35   35  35          1
## 3    32  32   32  32          1
## 4    15  15   15  15          1
## 5     3   3    3   3          1
## Inf  15  15   15  15          1
##
## Goodness-of-fit for model statistics
##
##                  obs  min    mean  max MC p-value
## edges             20   11   19.57   31          1
## nodecov.wealth 2168 1173 2119.66 3347          1
```

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


proportion of statistics
0.0
0.2
0.4
0.6
0.8
1.0
edges
nodecov.wealth
model statistics


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)
```

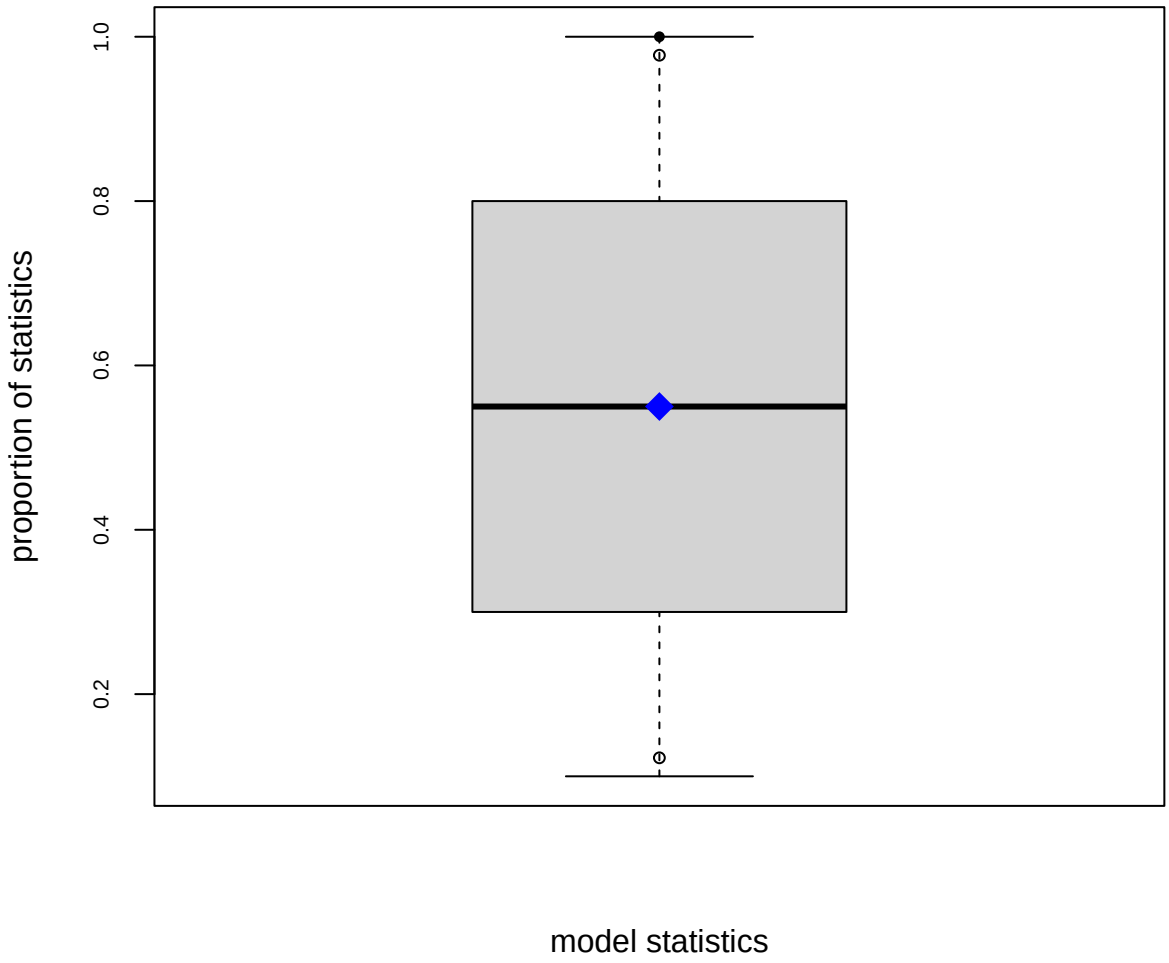
```
## Starting maximum pseudolikelihood estimation (MPLE):
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Finished MPLE.
## Stopping at the initial estimate.
## Evaluating log-likelihood at the estimate.
```

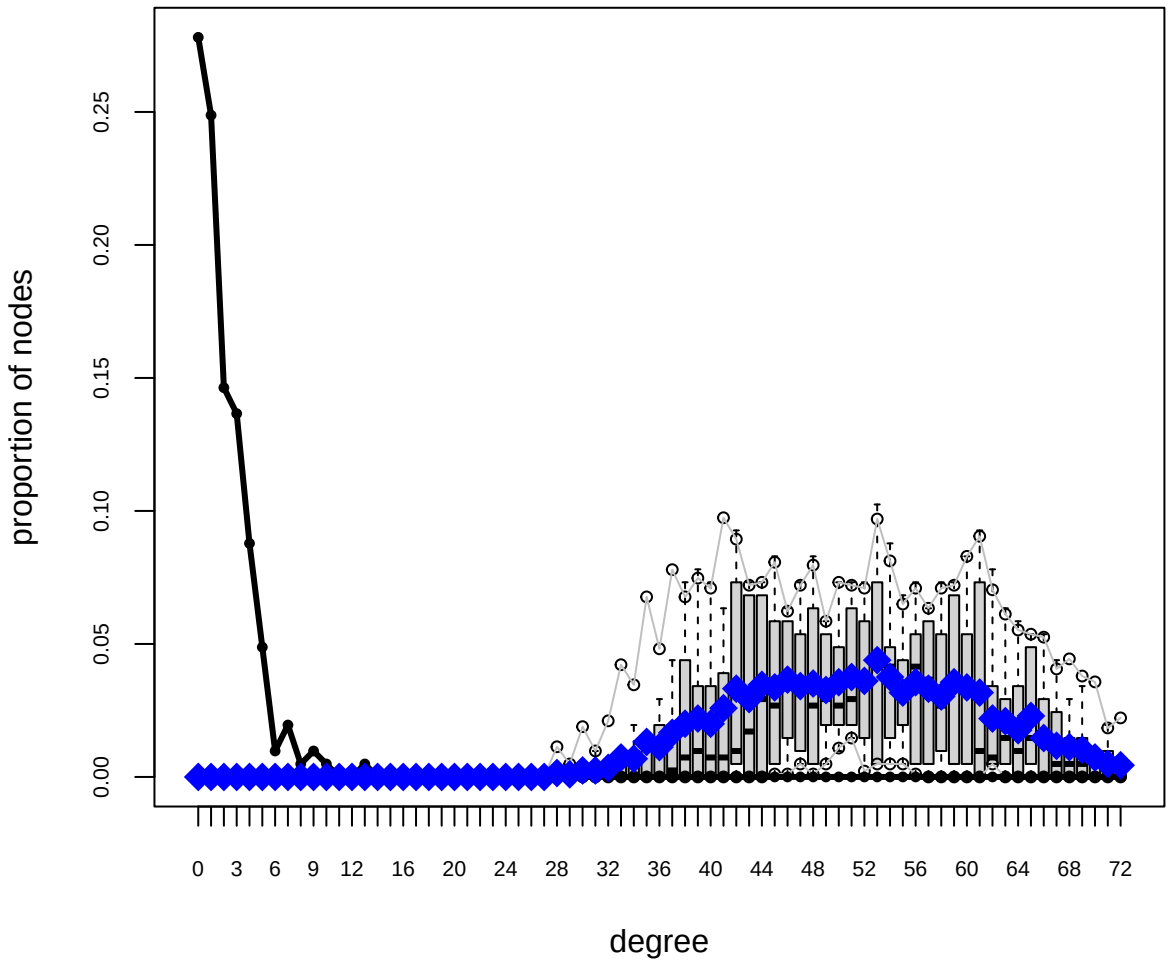
```
mesamodel.02.gof <- gof(mesamodel.02~degree + esp + distance,
                        control = control.gof.formula(nsim=10))
```

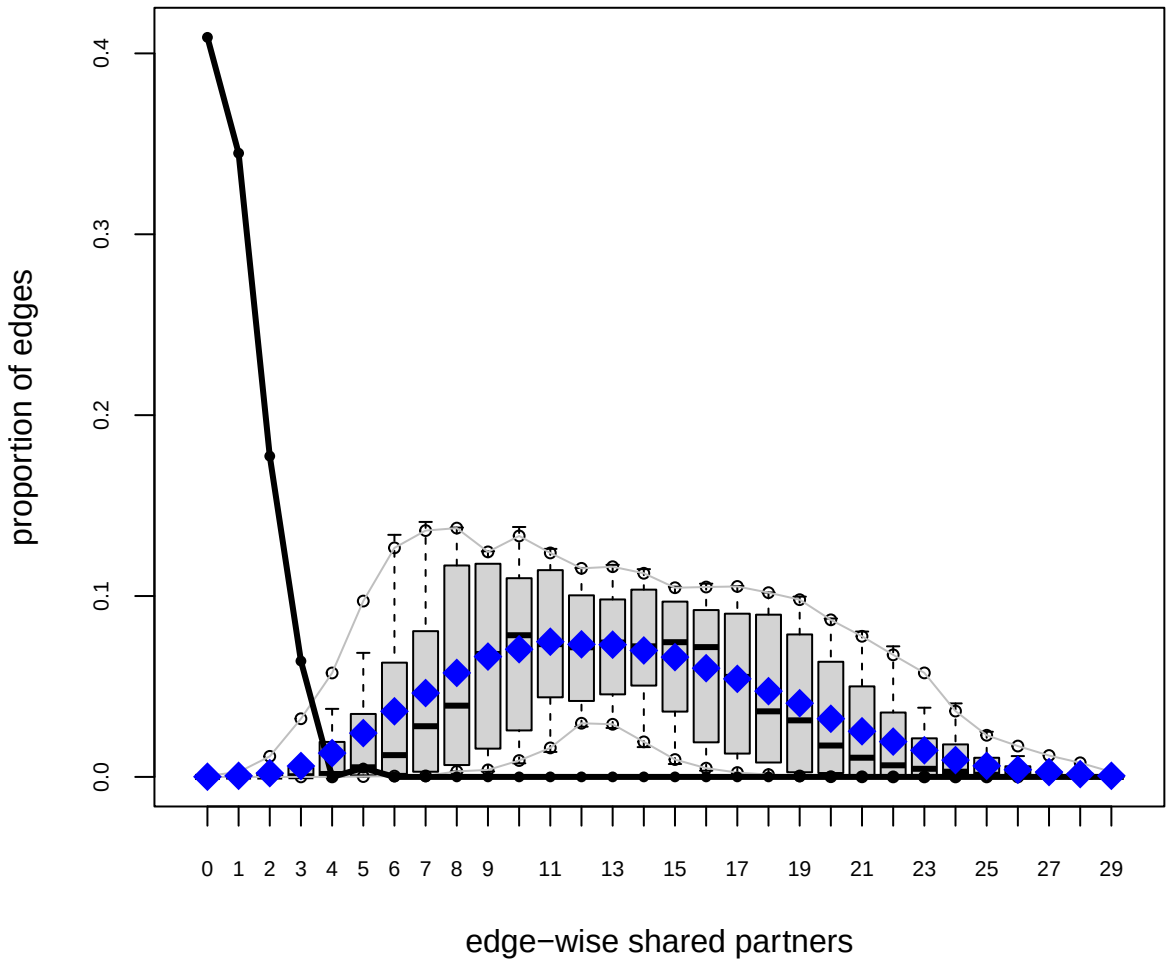
```
## Warning in check.control.class(c("gof.ergm", "gof.formula"), "gof.ergm"):
## Using control.gof.formula(...) as the control parameter of gof.ergm(...)
## is suboptimal and may overwrite some settings that should be preserved. Use
## control.gof.ergm(...) instead.
## Warning in gof.formula(object = object$formula, coef = coef, GOF = GOF, :
```

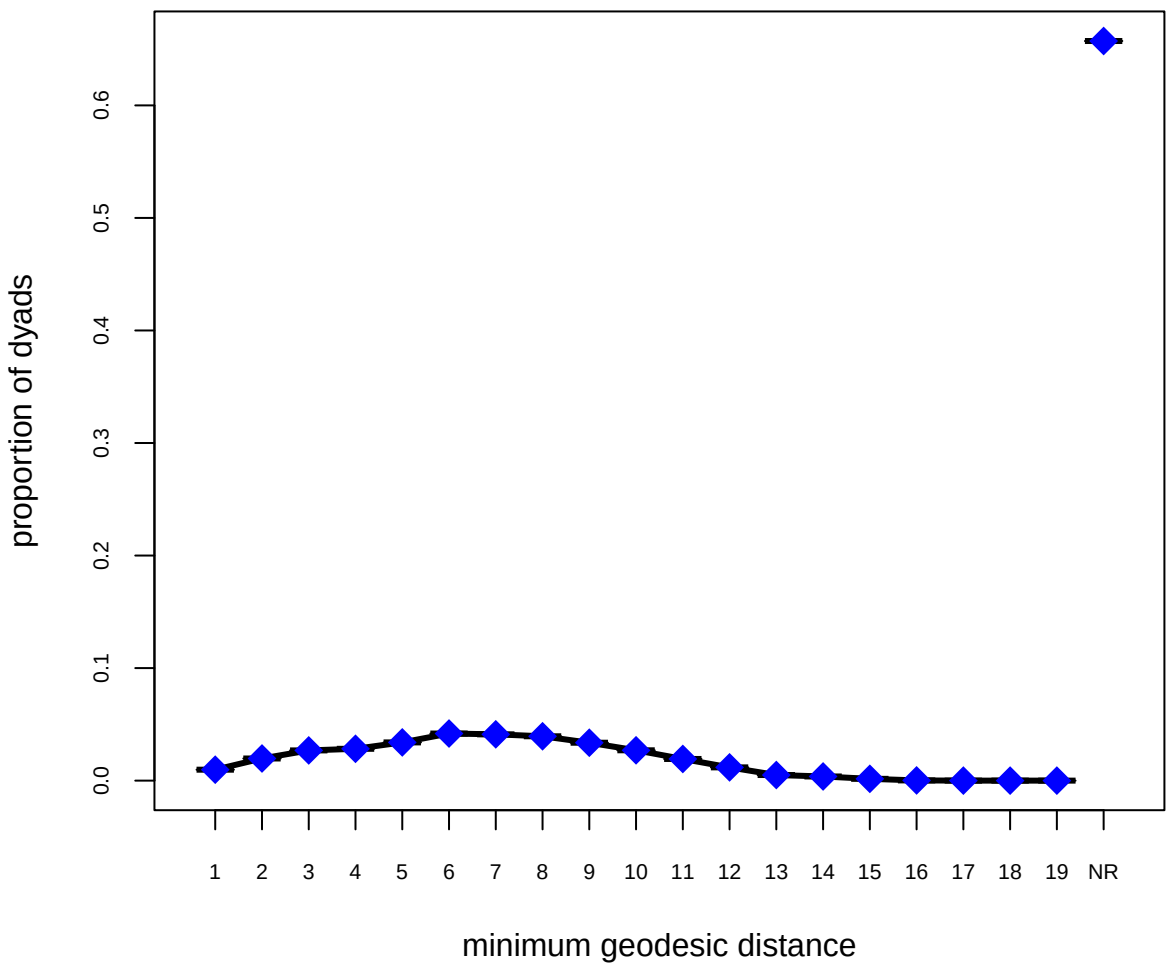
```
## No parameter values given, using 0.
```

```
plot(mesamodel.02.gof)
```




proportion of nodes
0.00
0.05
0.10
0.15
0.20
0.25
0 3 6 9 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
degree


proportion of edges
0.0
0.1
0.2
0.3
0.4
0 1 2 3 4 5 6 7 8 9 11 13 15 17 19 21 23 25 27 29
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

0.8 7. Diagnostics: troubleshooting and checking for model degeneracy

When a model is not a good representation of the observed network, the simulated networks produced in the MCMC chains 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. When this happens, it basically means the model you specified would not have produced the network you observed. Some models, we now know, would almost never produce an interesting network with this density – this is what we call "model degeneracy."

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

In that 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. We can, however, still 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.

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:

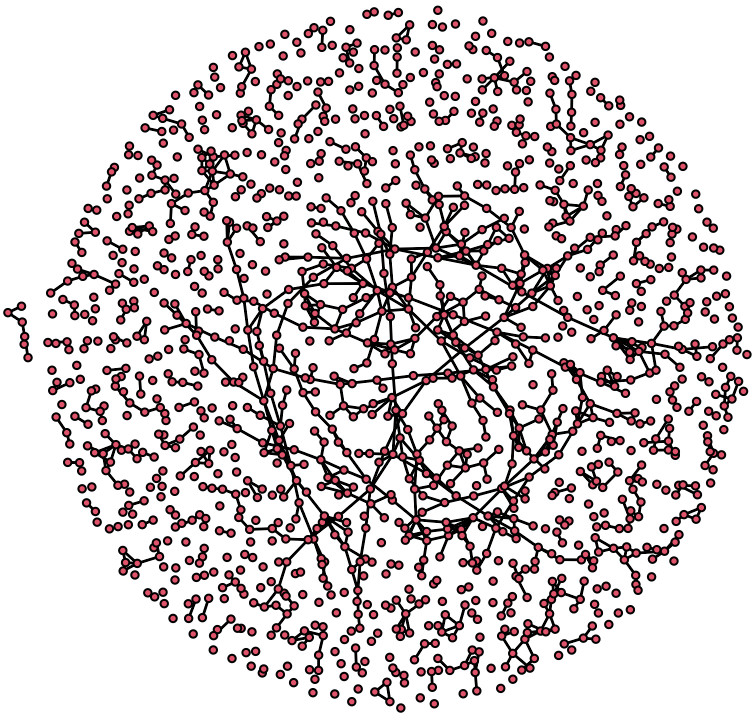
```
mcmc.diagnostics(fit, center=F)
```

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 60:
## Error in ergm.MCMLE(init, nw, model, initialfit = (initialfit <- NULL), : Number of edges in a simula
```

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)
```

One approach to solving model degeneracy would be to modify the parameters of MCMC algorithm. As one example, increasing the MCMC sample size can sometimes help:

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

But it does not solve the problem in this case, and in general this type of degeneracy is unlikely to be fixed by tuning the MCMC parameters.

How about trying a different model specification: use GWESP, the more robust approach to modeling triangles? (For a technical introduction to GWESP see Hunter 2007; for a more intuitive description and empirical application see Goodreau Kitts and Morris 2009 in the references.)

Note that we are running with a somewhat lower precision than the default, to save time.

```
fit <- ergm(magnolia~edges+gwesp(0.5,fixed=T)+nodematch('Grade')+nodematch('Race')+
  nodematch('Sex'),
 control =  control.ergm(seed=1, MCMLE.MCMC.precision=2))
mcmc.diagnostics(fit)
```

Still degenerate, though a bit better. Let's try adding some dyad-independent terms that can help to restrict the effects of the tie dependence.

One more try...

```
fit <- ergm(magnolia~edges+gwesp(0.25,fixed=T)+nodematch('Grade')+nodematch('Race')+
  nodematch('Sex'),
 control = control.ergm(seed=1, MCMLE.MCMC.precision=2),
 verbose=T)
```

```
## Evaluating network in model.
```

```
## Initializing unconstrained Metropolis-Hastings proposal: 'ergm:MH_TNT'.
## Initializing model...
## Model initialized.
## Using initial method 'MPLE'.
## Fitting initial model.
## Starting maximum pseudolikelihood estimation (MPLE):
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Finished MPLE.
## Starting Monte Carlo maximum likelihood estimation (MCMLE):
## Density guard set to 19563 from an initial count of 974 edges.
##
## Iteration 1 of at most 60 with parameter:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##       -9.6939224        1.6624034        2.8170436        0.9870588
##    nodematch.Sex
##        0.6188166
## Starting unconstrained MCMC...
## Back from unconstrained MCMC.
## New interval = 8192.
## Average estimating function values:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##        61.845324        -5.410138        55.258993        64.460432
##    nodematch.Sex
##         5.805755
## Starting MCMLE Optimization...
## Optimizing with step length 0.5407.
## Using lognormal metric (see control.ergm function).
## Using log-normal approx (no optim)
## The log-likelihood improved by 2.1895.
## Estimating equations are not within tolerance region.
##
## Iteration 2 of at most 60 with parameter:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##       -9.7175856        1.7379488        2.7830226        0.9134718
##    nodematch.Sex
##        0.6871173
## Starting unconstrained MCMC...
```

```
## Back from unconstrained MCMC.
## New interval = 4096.
## Average estimating function values:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##        -52.96930        -80.49239        -52.42105        -54.13158
##    nodematch.Sex
##        -67.34211
## Starting MCMLE Optimization...
## Optimizing with step length 0.7083.
## Using lognormal metric (see control.ergm function).
## Using log-normal approx (no optim)
## The log-likelihood improved by 2.8758.
## Distance from origin on tolerance region scale: 5.70768467515906 (previously 8.24874646183335).
## Estimating equations are not within tolerance region.
##
## Iteration 3 of at most 60 with parameter:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##       -9.8153266        1.8157845        2.7799571        0.9475458
##    nodematch.Sex
##        0.7665476
## Starting unconstrained MCMC...
## Back from unconstrained MCMC.
## New interval = 8192.
## Average estimating function values:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##        121.02867        112.63898        120.07527        116.84588
##    nodematch.Sex
##         87.23297
## Starting MCMLE Optimization...
## Optimizing with step length 0.5668.
## Using lognormal metric (see control.ergm function).
## Using log-normal approx (no optim)
## The log-likelihood improved by 2.0914.
## Distance from origin on tolerance region scale: 6.48561775303213 (previously 4.90905048523301).
## Estimating equations are not within tolerance region.
##
## Iteration 4 of at most 60 with parameter:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##       -9.7906060        1.7779830        2.7639442        0.9136299
##    nodematch.Sex
##        0.7639413
## Starting unconstrained MCMC...
## Back from unconstrained MCMC.
## New interval = 32768.
## Average estimating function values:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##        -25.96691        -30.84944        -23.84559        -22.23897
##    nodematch.Sex
##        -22.92647
## Starting MCMLE Optimization...
## Optimizing with step length 1.0000.
## Using lognormal metric (see control.ergm function).
## Using log-normal approx (no optim)
## Starting MCMC s.e. computation.
```

```
## The log-likelihood improved by 0.4106.
## Distance from origin on tolerance region scale: 0.409082365182927 (previously 4.87495501109811).
## Test statistic: T^2 = 16.3800103801879, with 5 free parameters and 103.272815859912 degrees of freedo
## Convergence test p-value: 0.0111. Not converged with 99% confidence; increasing sample size.
## 99% confidence critical value = 16.6826634575301.
##
## Iteration 5 of at most 60 with parameter:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##       -9.7868264        1.8092717        2.7511479        0.8999260
##    nodematch.Sex
##        0.7799796
## Starting unconstrained MCMC...
## Back from unconstrained MCMC.
## New interval = 16384.
## Average estimating function values:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##         20.76103         21.20549         20.48162         16.88603
##    nodematch.Sex
##         23.86029
## Starting MCMLE Optimization...
## Optimizing with step length 1.0000.
## Using lognormal metric (see control.ergm function).
## Using log-normal approx (no optim)
## Starting MCMC s.e. computation.
## The log-likelihood improved by 0.2802.
## Distance from origin on tolerance region scale: 0.279131529038502 (previously 0.438040084761353).
## Test statistic: T^2 = 9.2263929806771, with 5 free parameters and 71.1866036472601 degrees of freedom
## Convergence test p-value: 0.1371. Not converged with 99% confidence; increasing sample size.
## 99% confidence critical value = 17.4967741142578.
##
## Iteration 6 of at most 60 with parameter:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##       -9.7744387        1.7953558        2.7485904        0.9066707
##    nodematch.Sex
##        0.7555070
## Starting unconstrained MCMC...
## Back from unconstrained MCMC.
## New interval = 16384.
## Average estimating function values:
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##        -8.055666        -8.673639        -6.856859        -7.343936
##    nodematch.Sex
##        -7.001988
## Starting MCMLE Optimization...
## Optimizing with step length 1.0000.
## Using lognormal metric (see control.ergm function).
## Using log-normal approx (no optim)
## Starting MCMC s.e. computation.
## The log-likelihood improved by 0.0353.
## Distance from origin on tolerance region scale: 0.0352067466953044 (previously 0.357042940661241).
## Test statistic: T^2 = 39.1471728597526, with 5 free parameters and 185.728541539231 degrees of freedo
## Convergence test p-value: < 0.0001. Converged with 99% confidence.
## Finished MCMLE.
## Evaluating log-likelihood at the estimate. Initializing model to obtain the list of dyad-independent
```

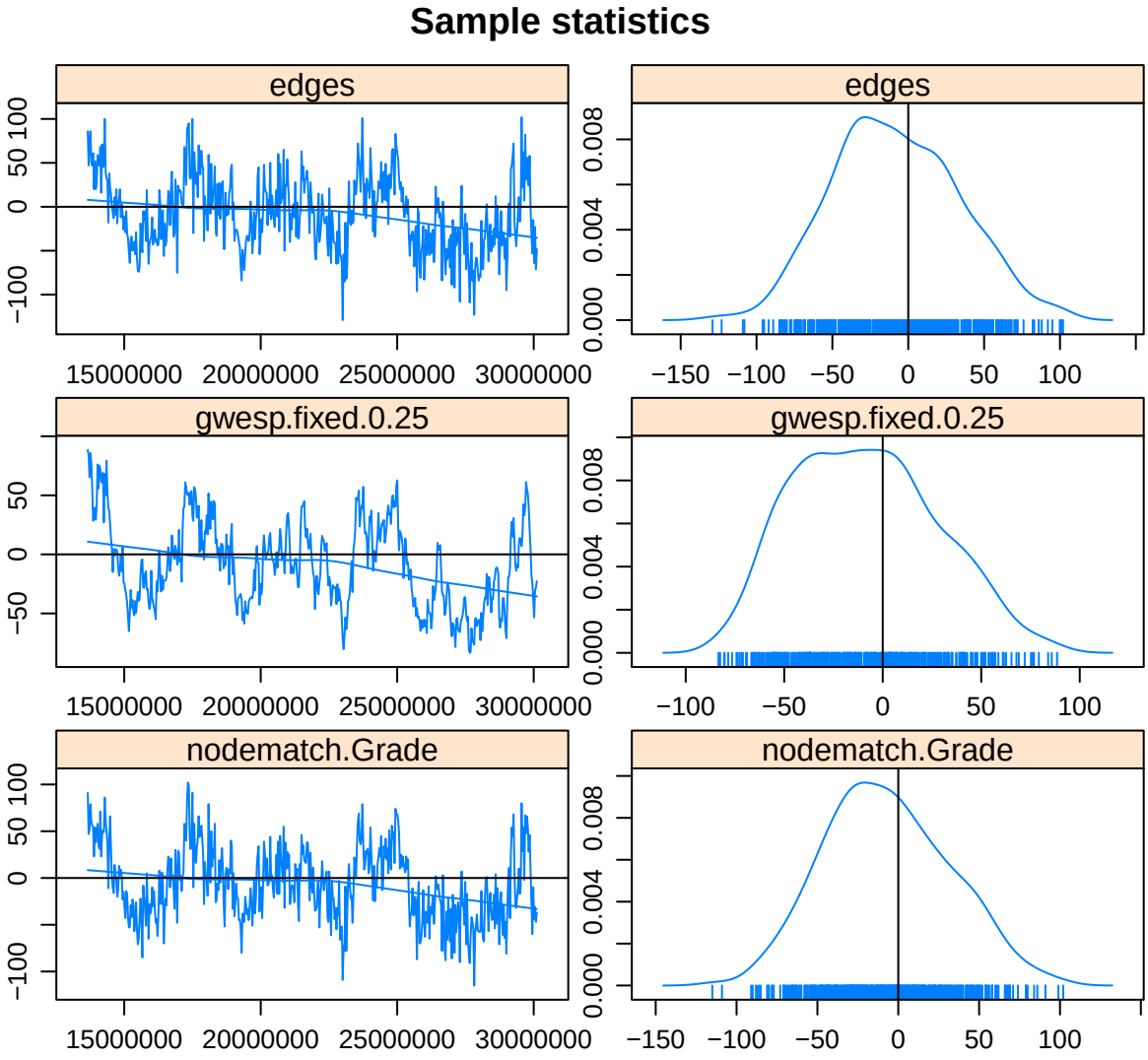
```
## Fitting the dyad-independent submodel...
## Dyad-independent submodel MLE has likelihood 732733.5 at:
## [1] -10.0127658   0.0000000   3.2310453   1.1964585   0.8843782   0.0000000
## Bridging between the dyad-independent submodel and the full model...
## Setting up bridge sampling...
## Initializing model and proposals...
## Model and proposals initialized.
## Using 16 bridges: Running theta=[-9.7826835, 1.7466178, 2.7575738, 0.9178091, 0.7648699, 0.0000000].
## Running theta=[-9.7975276, 1.6339328, 2.7881203, 0.9357865, 0.7725801, 0.0000000].
## Running theta=[-9.8123716, 1.5212477, 2.8186669, 0.9537639, 0.7802904, 0.0000000].
## Running theta=[-9.8272156, 1.4085627, 2.8492134, 0.9717412, 0.7880006, 0.0000000].
## Running theta=[-9.8420596, 1.2958777, 2.8797600, 0.9897186, 0.7957108, 0.0000000].
## Running theta=[-9.856904, 1.183193, 2.910307, 1.007696, 0.803421, 0.000000].
## Running theta=[-9.8717477, 1.0705077, 2.9408531, 1.0256734, 0.8111312, 0.0000000].
## Running theta=[-9.8865917, 0.9578227, 2.9713996, 1.0436508, 0.8188414, 0.0000000].
## Running theta=[-9.9014357, 0.8451376, 3.0019462, 1.0616281, 0.8265516, 0.0000000].
## Running theta=[-9.9162797, 0.7324526, 3.0324927, 1.0796055, 0.8342618, 0.0000000].
## Running theta=[-9.9311237, 0.6197676, 3.0630393, 1.0975829, 0.8419721, 0.0000000].
## Running theta=[-9.9459678, 0.5070826, 3.0935858, 1.1155603, 0.8496823, 0.0000000].
## Running theta=[-9.9608118, 0.3943976, 3.1241324, 1.1335377, 0.8573925, 0.0000000].
## Running theta=[-9.9756558, 0.2817125, 3.1546789, 1.1515151, 0.8651027, 0.0000000].
## Running theta=[-9.9904998, 0.1690275, 3.1852255, 1.1694924, 0.8728129, 0.0000000].
## Running theta=[-10.00534383,  0.05634251,  3.21577201,  1.18746983,  0.88052312,  0.00000000].
## .
## Bridge sampling finished. Collating...
## Bridging finished.
## This model was fit using MCMC.  To examine model diagnostics and
## check for degeneracy, use the mcmc.diagnostics() function.
```

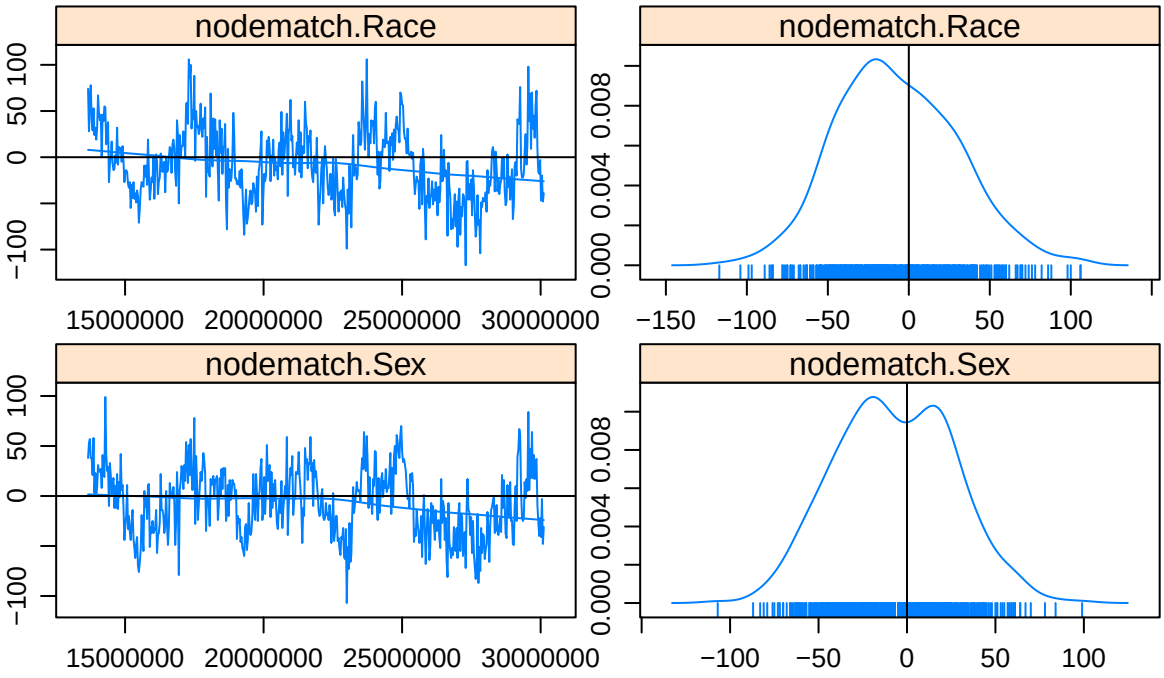
```
mcmc.diagnostics(fit)
```

```
## Sample statistics summary:
##
## Iterations = 13664256:30113792
## Thinning interval = 32768
## Number of chains = 1
## Sample size per chain = 503
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                     Mean    SD Naive SE Time-series SE
## edges            -8.056 41.74    1.861          8.824
## gwesp.fixed.0.25 -8.674 36.03    1.607          8.292
## nodematch.Grade  -6.857 38.99    1.739          7.600
## nodematch.Race   -7.344 37.50    1.672          8.131
## nodematch.Sex    -7.002 33.16    1.479          7.250
##
## 2. Quantiles for each variable:
##
##                     2.5%    25%    50%   75% 97.5%
## edges            -83.45 -37.00 -11.00 20.50 72.00
## gwesp.fixed.0.25 -67.69 -37.09 -11.05 15.85 61.35
## nodematch.Grade  -78.00 -33.50 -10.00 20.50 69.00
## nodematch.Race   -75.45 -34.00 -10.00 19.00 69.45
```

```
## nodematch.Sex    -66.90 -31.50  -9.00 17.00 58.45
##
##
## Are sample statistics significantly different from observed?
##                 edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
## diff.      -8.0556660       -8.6736385      -6.8568588     -7.3439364
## test stat. -0.9129511       -1.0459691      -0.9022062     -0.9031789
## P-val.      0.3612683        0.2955753       0.3669473      0.3664310
##            nodematch.Sex Overall (Chi^2)
## diff.         -7.0019881              NA
## test stat.    -0.9657517       5.0949851
## P-val.         0.3341685       0.4209689
##
## Sample statistics cross-correlations:
##                      edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
## edges            1.0000000        0.8106516       0.9491382      0.9303179
## gwesp.fixed.0.25 0.8106516        1.0000000       0.8271524      0.7831491
## nodematch.Grade  0.9491382        0.8271524       1.0000000      0.8885677
## nodematch.Race   0.9303179        0.7831491       0.8885677      1.0000000
## nodematch.Sex    0.8946083        0.7378557       0.8528858      0.8263070
##                  nodematch.Sex
## edges                0.8946083
## gwesp.fixed.0.25     0.7378557
## nodematch.Grade      0.8528858
## nodematch.Race       0.8263070
## nodematch.Sex        1.0000000
##
## Sample statistics auto-correlation:
## Chain 1
##                edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
## Lag 0      1.0000000        1.0000000       1.0000000      1.0000000
## Lag 32768  0.6178933        0.9145207       0.6710198      0.6405492
## Lag 65536  0.5859769        0.8501900       0.6158360      0.6008631
## Lag 98304  0.5568072        0.7950794       0.5888809      0.5705985
## Lag 131072 0.5307569        0.7379496       0.5729138      0.5749957
## Lag 163840 0.4827337        0.6862026       0.5272875      0.5294008
##            nodematch.Sex
## Lag 0          1.0000000
## Lag 32768      0.6314127
## Lag 65536      0.5906851
## Lag 98304      0.5456017
## Lag 131072     0.5105355
## Lag 163840     0.4667924
##
## 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
##            1.545            1.420            1.036            1.432
##    nodematch.Sex
##            1.725
```

```
##
## Individual P-values (lower = worse):
##            edges gwesp.fixed.0.25  nodematch.Grade   nodematch.Race
##       0.12245021       0.15561204       0.30040018       0.15222188
##    nodematch.Sex
##       0.08445473
## Joint P-value (lower = worse):  0.7227366 .
```






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

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

Degeneracy is often 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.

0.9 8. 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.

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 http://statnet.org/movies

While the `ergm` package can be used with egocentric data, we recommend instead to use the package `ergm.ego`. This package includes accurate statistical inference and many utilities that simplify the task of reading in the data, conducting exploratory analyses, calculating the sample "target statistics", and specifying model options.

We have a workshop/tutorial for `ergm.ego` at the statnet Workshops wiki.

0.10 9. Additional functionality in statnet and other package

`statnet` is a suite of packages that are designed to work together, and provide tools for a wide range of different types of network data analysis. Examples include temporal network models and dynamic network vizualizations, multilevel network modeling, latent cluster models and network diffusion and epidemic models. Development is ongoing, and there are new packages, and new functionality added to existing packages on a regular basis.

All of these packages can be downloaded from CRAN. For more detailed information, please visit the `statnet` webpage www.statnet.org.

0.10.1 Current statnet packages

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

- `sna` – classical social network analysis utilities
- `tsna` – descriptive statistics for temporal network data
- `tergm` – temporal ergms for dynamic networks
- `ergm.ego`– estimation/simulation of ergms from egocentrically sampled data
- `ergm.count` – models for tie count network data
- `ergm.rank` – models for tie rank network data
- `relevent` – relational event models for networks
- `latentnet` – latent space and latent cluster analysis
- `degreenet` – MLE estimation for degree distributions (negative binomial, Poisson, scale-free, etc.)
- `networksis` – simulation of bipartite networks with given degree distributions
- `ndtv` package – network movie maker
- `EpiModel` – network modeling of infectious disease and social diffusion processes

0.10.2 Additional functionality in base `ergm`

- ERGMs for valued ties

0.10.3 Extensions by other developers

There are now a number of excellent packages developed by others that extend the functionality of statnet. The easiest way to find these is to look at the "reverse depends" of the `ergm` package on CRAN. Examples include:

- `Bergm` – Bayesian Exponential Random Graph Models
- `btergm` – Temporal Exponential Random Graph Models by Bootstrapped Pseudolikelihood
- `hergm` – hierarchical ERGMs for multi-level network data
- `xergm` – extensions to ERGM modeling

0.10.4 The `statnet` development team

Pavel N. Krivitsky pavel@uow.edu.au Martina Morris morrism@u.washington.edu
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
Samuel M. Jenness samuel.m.jenness@emory.edu

0.11 References

The best place to start is the special issue of the *Journal of Statistical Software* (JSS) devoted to `statnet`: link

The nine papers in this issue cover a wide range of theoretical and practical topics related to ERGMs, and their implementation in `statnet`.

HOWEVER: Note that this issue was written in 2008. The statnet code base has evolved considerably since that time, so some of the syntax specified in the articles may no longer work (in most cases because it has been replace with something better).

An overview of most recent update, `ergm 4`, can be found in here: link.

For social scientists, a good introductory application paper is:

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

Dealing with Model Degeneracy

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

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

Hunter, D. R. (2007). Curved Exponential Family Models for Social Networks. *Social Networks*, 29(2), 216-230.link

Temporal ERGMs

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

Egocentric ERGMS

Krivitsky, P. N., & Morris, M. (2017). Inference for social network models from egocentrically sampled data, with application to understanding persistent racial disparities in HIV prevalence in the US. *Annals of Applied Statistics*, 11(1), 427-455.link