Exponential Random Graph Models (ERGMs) using statnet

Statnet Development Team

Contents

Last updated 2021-06-28

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

Pavel N. Krivitsky (University of New South Wales)
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)
Chad Klumb (University of Washington)
Skye Bender de-Moll (Oakland, CA)

The network modeling software demonstrated in this tutorial is authored by Pavel Krivitsky (`ergm`) and Carter Butts (`network` and `sna`).

The `statnet` Project

All `statnet` packages are open-source, written for the **R** computing environment, and published on CRAN. The source repositories are hosted on GitHub. Our website is statnet.org

- Need help? For general questions and comments, please email the statnet users group at statnet_help@uw.edu. You'll need to join the listserv if you're not already a member. You can do that here: statnet_help listserve.
- Found a bug in our software? Please let us know by filing an issue in the appropriate package GitHub repository, with a reproducible example.

- Want to request new functionality? We welcome suggestions – you can make a request by filing an issue on the appropriate package GitHub repository. The chances that this functionality will be developed are substantially improved if the requests are accompanied by some proposed code (we are happy to review pull requests).
- For all other issues, please email us at contact@statnet.org.

Introduction to this workshop/tutorial.

This workshop and tutorial provide an introduction to statistical modeling of network data with *Exponential family Random Graph Models* (ERGMs) using `statnet` software. This online tutorial is also designed for self-study, with example code and self-contained data. The `statnet` packages we will be demonstrating are:

- `network` – storage and manipulation of network data
- `ergm` – statistical tools for estimating ERGMs, model assessment, and network simulation.

The `ergm` package has more advanced functionality that is not covered in this workshop. An overview can be found in this preprint.

Prerequisites

This workshop assumes basic familiarity with **R**, experience with network concepts, terminology and data, and familiarity with the general framework for statistical modeling and inference. While previous experience with ERGMs is not required, some of the topics covered here may be difficult to understand without a strong background in linear and generalized linear models in statistics.

Software installation

Minimally, you will need to install the latest version of **R** (available here) and the `statnet` packages `ergm` and `network` to run the code presented here (`ergm` will automatically install `network` when it is loaded). The workshops are conducted using the free version of `Rstudio` (available here).

The full set of installation intructions with details can be found on the `statnet` workshop wiki.

If you have not already downloaded the `statnet` packages for this workshop, the quickest way to install these (and the other most commonly used packages from the `statnet` suite), is to open an R session and type:

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

```
library(ergm)
```

```
Loading required package: network
```

```
'network' 1.17.1 (2021-06-12), part of the Statnet Project
* 'news(package="network")' for changes since last version
* 'citation("network")' for citation information
* 'https://statnet.org' for help, support, and other information
```

```
'ergm' 4.0.1 (2021-06-20), part of the Statnet Project
* 'news(package="ergm")' for changes since last version
* 'citation("ergm")' for citation information
* 'https://statnet.org' for help, support, and other information
```

```
'ergm' 4 is a major update that introduces some backwards-incompatible
changes. Please type 'news(package="ergm")' for a list of major
changes.
```

You can check the version number with:

```
packageVersion("ergm")
```

```
[1] '4.0.1'
```

Throughout, we will set a random seed via `set.seed()` for commands in tutorial that require simulating random values—this is not necessary, but it ensures that you will get the same results as the online tutorial.

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 `statnet` software tools. 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 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:

- Define a model for a network that includes covariates representing features like 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 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.

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 (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 (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 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, 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.

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')
```

```
starting httpd help server ... done
```

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.

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. We will see exactly how this works in the sections that follow.

Loading network data

Network data can come in many different forms — ties can be stored as edgelists or sociomatrices in .csv files, or as exported data from other programs like Pajek; attributes for the nodes, ties and dyads can also come in various forms. All can be read into *R* using either standard methods (for .csv files), or methods from the `network` package. For more information:

```
?read.paj
?read.paj.simplify
?loading.attributes
```

However you read them in, the data will need to be transformed into a `network` object — the format that statnet uses to store and work with network data. For information on how to do this:

```
?network
```

The `ergm` package also contains several network data sets, and we will use those here for demonstration purposes.

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

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.

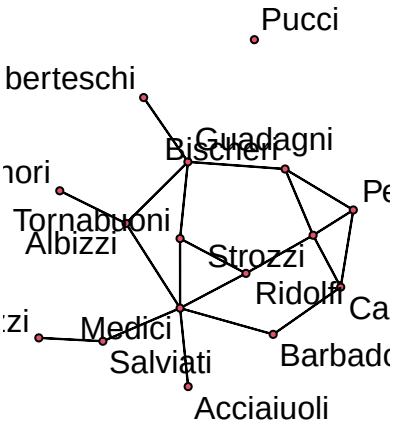
```
set.seed(123) # The plot.network function uses random values
data(florentine) # loads flomarriage and flobusiness data
flomarriage # Look at the flomarriage network properties (uses `network`), esp. the vertex attributes
```

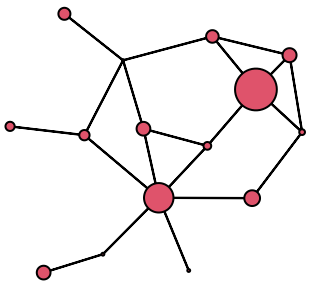
```
 Network attributes:
  vertices = 16
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 20
    missing edges= 0
    non-missing edges= 20

 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
```





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:

$$\text{my.network} \sim \text{my.vector.of.model.terms}$$

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 run a `summmary` command on any 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.

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:

$$\begin{aligned}&= \exp(-1.61)/(1+\exp(-1.61)) \\ &= 0.167\end{aligned}$$

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 $20/120 = 0.167$.

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, `ergm` automatically uses its stochastic MCMC-based estimation algoirthm, so your results may differ slightly unless you use the same `set.seed` value:

```
set.seed(321)
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     -1.6930     0.3795      0  -4.461   <1e-04 ***
triangle   0.2038     0.6144      0   0.332     0.74
---
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. = 0.01179)
```

Now, how should we interpret coefficients?

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

$$-1.69 \times \text{change in the number of ties} + 0.204 \times \text{change in number of triangles.}$$

- For a tie that will create no triangles, the conditional log-odds is: -1.69.
- if one triangle: $-1.69 + 0.204 = -1.49$
- if two triangles: $-1.69 + 2 \times 0.204 = -1.29$
- the corresponding probabilities are shown here (note the use of the `plogis` and `coef` functions):

```
plogis(coef(flomodel.02)[[1]] + (0:2) * coef(flomodel.02)[[2]])
```

```
[1] 0.1553869 0.1840506 0.2166455
```

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"      "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] "call"            "ergm_version"    "MPLE_is_MLE"     "formula"
[25] "nw.stats"        "constrained"     "constraints"     "obs.constraints"
[29] "reference"       "estimate"        "estimate.desc"   "offset"
[33] "drop"            "estimable"       "null.lik"        "mle.lik"
```

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

```
     edges   triangle
-1.6929606  0.2038189
```

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? 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.

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

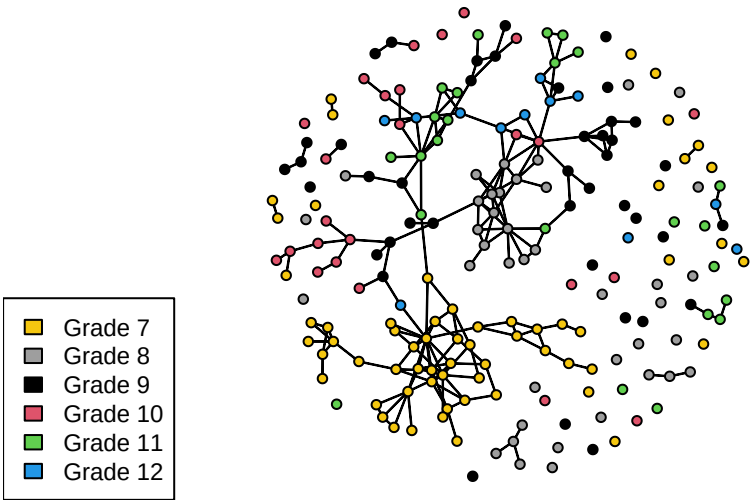
Nodal covariates: Homophily Let's try a larger network, a simulated mutual friendship network based on one of the schools from the AddHealth study. Here, we'll examine the homophily in friendships by grade and race. Both are discrete attributes so we use the ergm-term ***nodematch***.

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

```
set.seed(1)
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 va

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
```

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.

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-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.

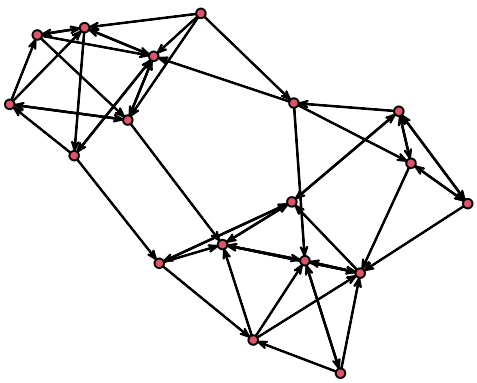
```
set.seed(2)
data(samplk)
ls() # directed data: Sampson's Monks
```

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

```
set.seed(3)
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   -2.1580     0.2233      0  -9.662   <1e-04 ***
mutual   2.3167     0.4848      0   4.779   <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: 267.5  on 304  degrees of freedom

AIC: 271.5  BIC: 279  (Smaller is better. MC Std. Err. = 0.3358)
```

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).

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.

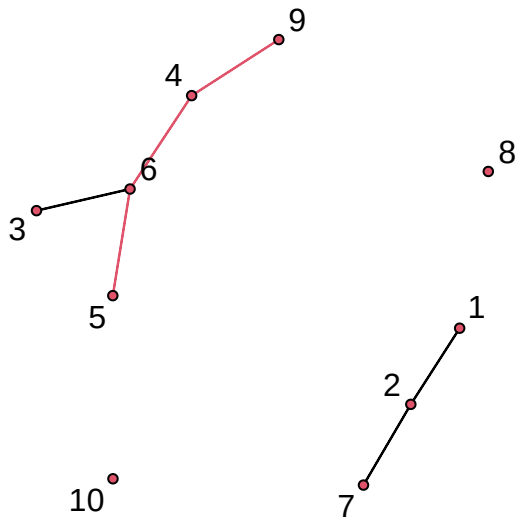
Start by estimating an ergm on a network with two missing ties, where both ties are identified as missing.

```
set.seed(4)
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.

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.

Terms provided with ergm

The `ergm` package provides myriad terms, and it can be difficult to absorb the full array of available model terms in any one place. This is particularly true with the release of `ergm` version 4.0, which expands the user's ability to create terms even further, for example through the use of operators. As mentioned above in Section 1, it is possible to search for specific topics using `search.ergmTerms` or to obtain help on the full array of terms using `help("ergm-terms")`.

The list of all terms is quite lengthy, so it may be helpful to start with a more concise list such as the one found here. A more detailed discussion can be found in volume 24, issue 4 of the *Journal of Statistical Software*.

To appreciate the expanded capabilities of the `ergm` package as of the release of version 4.0, we recommend Krivitsky et al (2021). In this article, Section 3 describes the enhanced flexibility to create specialized model terms involving functions of nodal covariates, and Section 4 explains how operators further extend the types of terms at the user's disposal.

Coding new ergm-terms

There is a `statnet` package, `ergm.userterms`, that provides the utilities needed to write 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.

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.

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. This approach basically works as follows:

- start with an initial vector of parameter values; the default is to use the maximum psuedo-likelihood estimates (MPLEs, using the logistic regression estimation algorithm)
- choose a dyad at random, and flip a coin, weighted by the model, to decide whether there will be a tie.
- repeat this for 1024 steps (the default "MCMC interval")
- take the network at the last step and calculate the network statistics (the ones in the model)
- repeat this 1024 times (the default "MCMC sample size")
- calculate the average for each statistic in the sample, and compare this to the observed statistic
- update the parameter estimates as needed
- repeat until the process converges: the difference between the MCMC sample average and the observed statistic is sufficiently small

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.

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.

```
set.seed(314159)
summary(flobusiness~edges+degree(1))
```

```
  edges degree1
     15       3
```

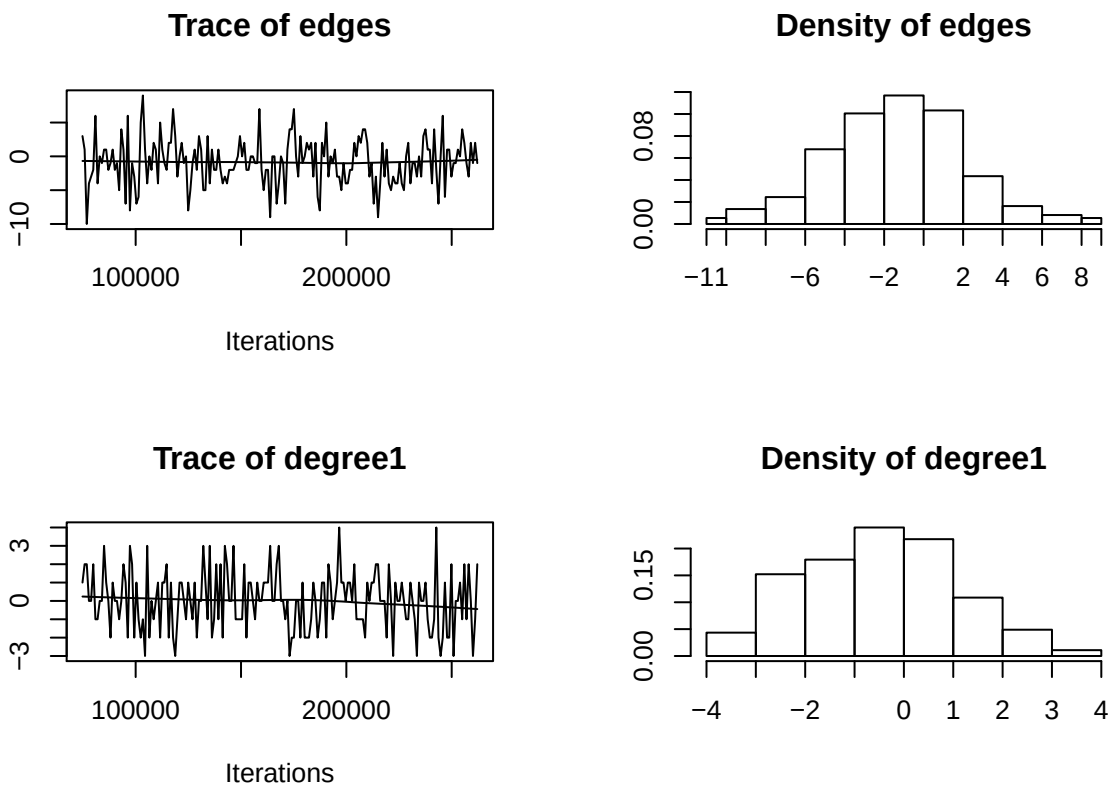
```
fit <- ergm(flobusiness~edges+degree(1))
summary(fit)
```

```
Call:
ergm(formula = flobusiness ~ edges + degree(1))

Monte Carlo Maximum Likelihood Results:

        Estimate Std. Error MCMC % z value Pr(>|z|)
edges    -2.0864     0.3172      0  -6.577   <1e-04 ***
degree1  -0.7273     0.6926      0  -1.050    0.294
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 166.4  on 120  degrees of freedom
 Residual Deviance:  89.4  on 118  degrees of freedom

AIC: 93.4  BIC: 98.97  (Smaller is better. MC Std. Err. = 0.03691)
```

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

```
Sample statistics summary:

Iterations = 74752:262144
Thinning interval = 1024
Number of chains = 1
Sample size per chain = 184

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

            Mean    SD Naive SE Time-series SE
edges   -0.86957 3.424   0.2524         0.2524
degree1  0.01087 1.568   0.1156         0.1156

2. Quantiles for each variable:

        2.5% 25% 50% 75% 97.5%
```

```
edges     -8  -3  -1   1     6
degree1   -3  -1   0   1     3


Are sample statistics significantly different from observed?
                   edges    degree1 Overall (Chi^2)
diff.      -0.8695652174 0.01086957              NA
test stat. -3.4453724178 0.09402663    13.498801538
P-val.      0.0005702731 0.92508800     0.001541427

Sample statistics cross-correlations:
             edges    degree1
edges    1.0000000 -0.3779053
degree1 -0.3779053  1.0000000

Sample statistics auto-correlation:
Chain 1
                 edges      degree1
Lag 0     1.0000000000  1.000000000
Lag 1024  0.1027448450 -0.024421732
Lag 2048 -0.0004618221 -0.006619202
Lag 3072  0.0847690361  0.095583540
Lag 4096 -0.0532426258  0.075582311
Lag 5120 -0.0241213922  0.031183769

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

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

  edges degree1
-0.1917  1.7736

Individual P-values (lower = worse):
     edges    degree1
0.84800704 0.07613595
Joint P-value (lower = worse):  0.2966402 .
```




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

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: the MCMC sample statistics are varying randomly around the observed values at each step (so the chain is "mixing" well), the sampled values show little serial correlation (so they are independent draws from the graph space) 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(snctrl)"), 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:

```
set.seed(271828)
fit <- ergm(flobusiness~edges+degree(1),
            control=snctrl(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.

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.

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.

We use simulations from the model as a way to assess its "goodness of fit", and that assessment should be done before using simulation for other purposes. But we will take a quick look here at how the simulation function works.

The `simulate` command is easy to run:

```
set.seed(101)
flomodel.03.sim <- simulate(flomodel.03,nsim=10)

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

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

```
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,]    17           1539
 [2,]    18           1742
 [3,]    21           2471
 [4,]    16           1304
 [5,]    18           1779
 [6,]    26           3143
 [7,]    22           2239
 [8,]    26           2905
 [9,]    24           2792
[10,]    15           1682
attr(,"monitored")
[1] FALSE FALSE
```

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

```
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,]    17           1539
 [2,]    18           1742
 [3,]    21           2471
 [4,]    16           1304
 [5,]    18           1779
 [6,]    26           3143
 [7,]    22           2239
 [8,]    26           2905
 [9,]    24           2792
[10,]    15           1682
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: 0x7f7fa6df7e40>

$reference
~Bernoulli
<environment: 0x7f7fa6df2a80>
```

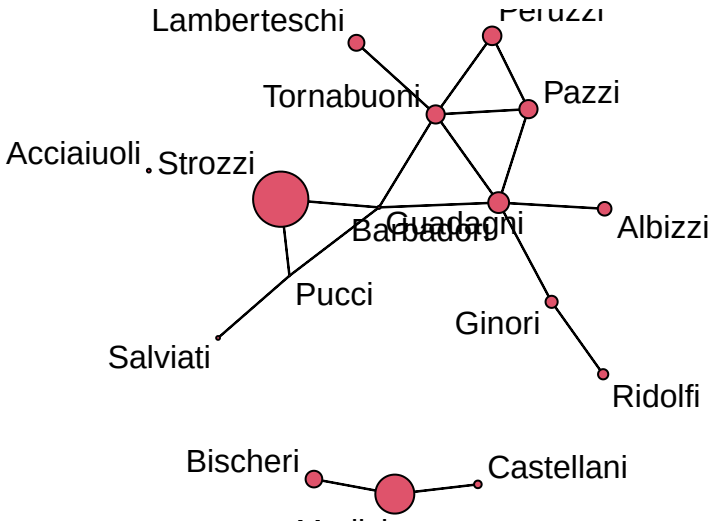
```
# are the simulated stats centered on the observed stats?
rbind("obs"=summary(flomarriage~edges+nodecov("wealth")),
      "sim mean"=colMeans(attr(flomodel.03.sim, "stats")))
```

```
         edges nodecov.wealth
obs       20.0         2168.0
sim mean  20.3         2159.6
```

```
# 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= 17
    missing edges= 0
    non-missing edges= 17

 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)
```



Voil'a. Of course, yours will look somewhat different.

Simulation from a model is a very powerful tool for examining the range of variation that can 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 plays a large role in analyzing egocentrically sampled data, 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.

Next, we will examine the primary use of simulation in the ergm package: we simulate networks from the fitted model to evaluate goodness of fit to the observed network.

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.

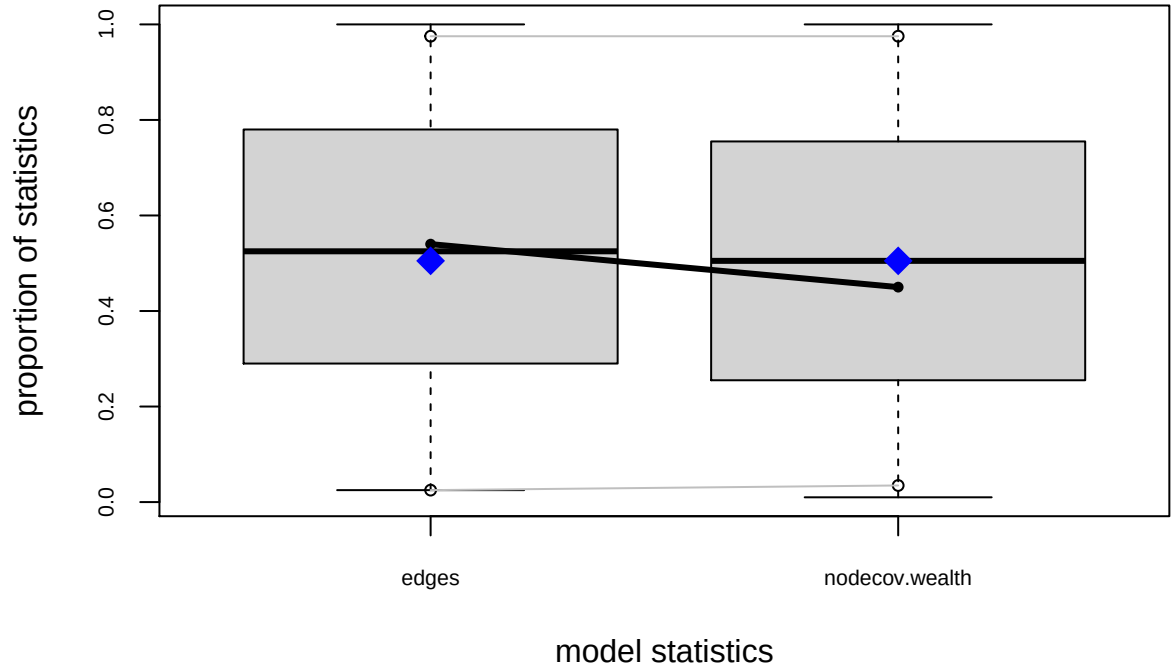
The **gof** function is a bit different than the **summary**, **ergm**, and **simulate** functions, in that it currently only takes 3 ergm-terms as arguments: degree, esp (edgwise share partners), and distance (geodesic distances). Each of these terms captures an aggregate network distribution, at either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

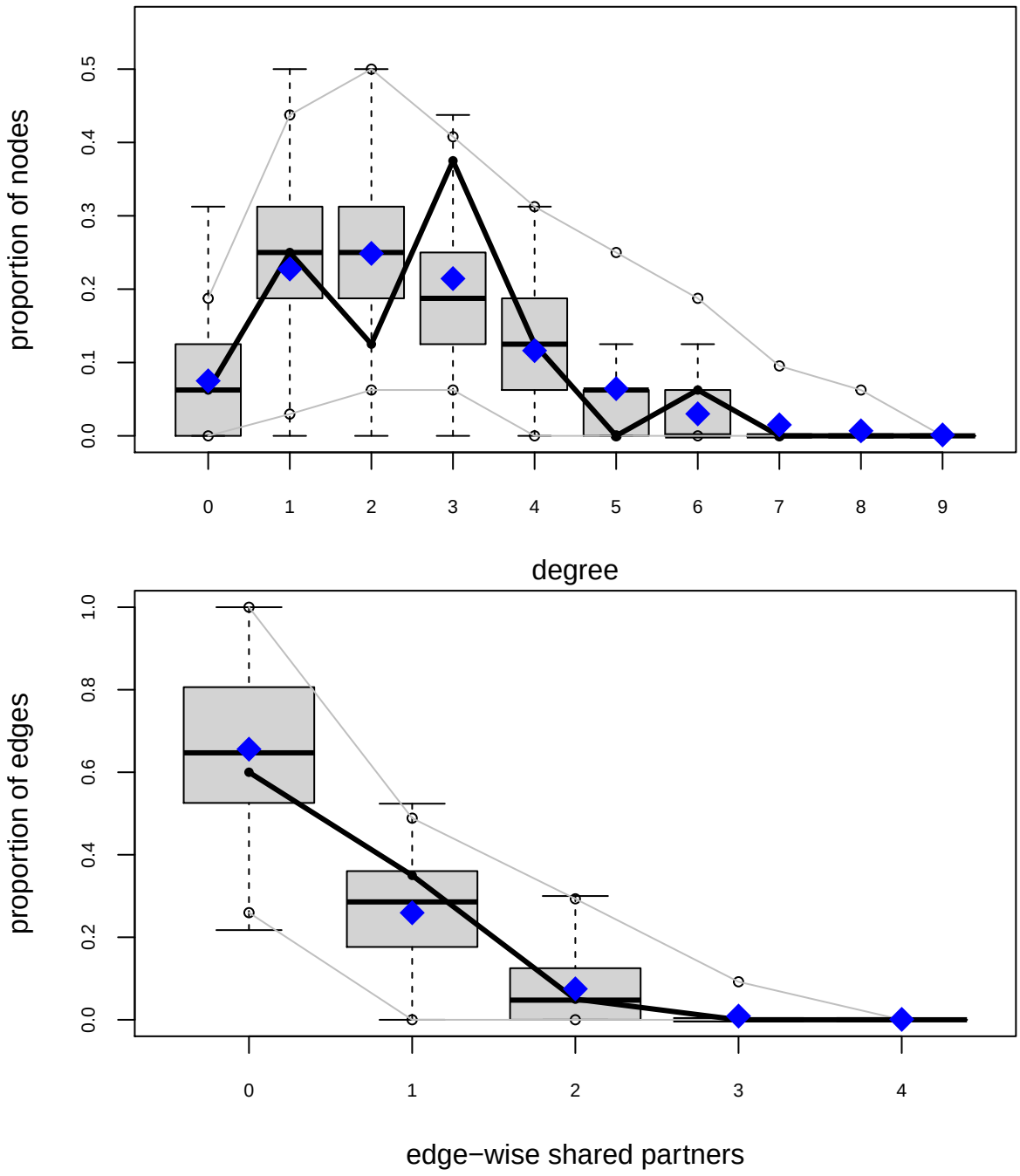
```
set.seed(54321) # The gof function uses random values
flomodel.03.gof <- gof(flomodel.03)
flomodel.03.gof
```

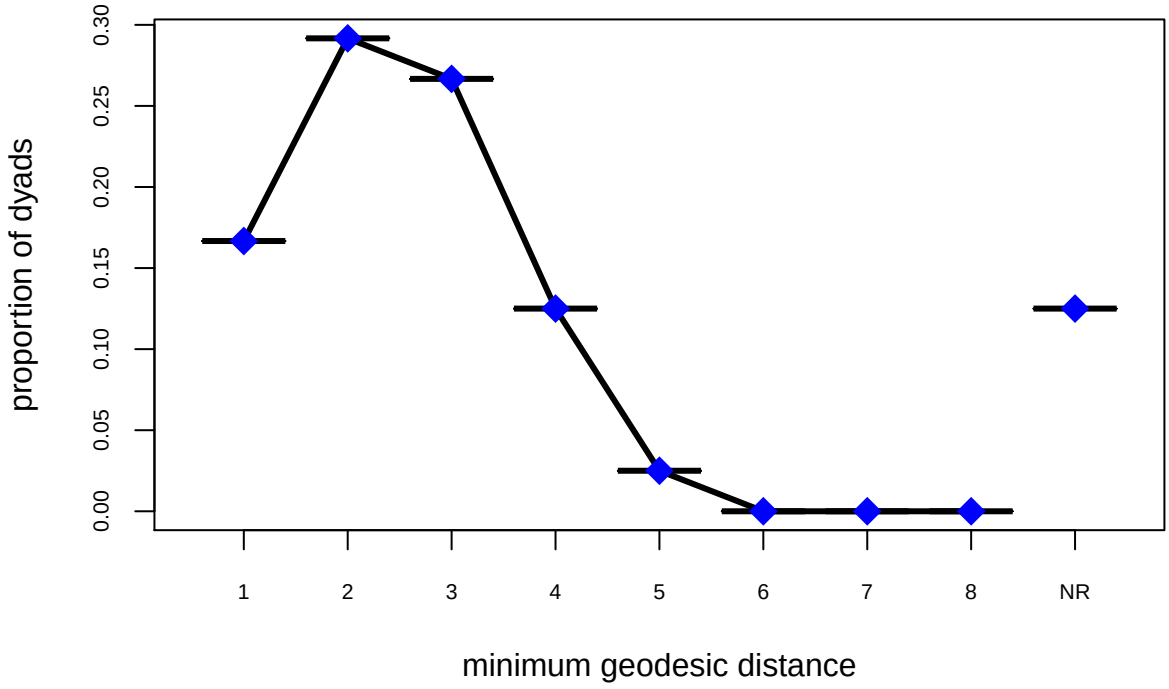
```
Goodness-of-fit for degree

         obs min mean max MC p-value
degree0    1   0 1.20   5       1.00
degree1    4   0 3.64   8       1.00
degree2    2   0 3.98   9       0.44
degree3    6   0 3.43   7       0.20
degree4    2   0 1.86   7       1.00
degree5    0   0 1.03   5       0.68
degree6    1   0 0.48   4       0.70
degree7    0   0 0.24   2       1.00
degree8    0   0 0.11   1       1.00
degree9    0   0 0.02   1       1.00
degree10   0   0 0.01   1       1.00

Goodness-of-fit for edgewise shared partner

     obs min  mean max MC p-value
esp0  12   5 12.65  19       0.86
esp1   7   0  5.49  15       0.72
esp2   1   0  1.71   8       1.00
esp3   0   0  0.22   5       1.00
esp4   0   0  0.03   2       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   13   20.10   37        1.0
nodecov.wealth 2168 1287 2201.89 3467        0.9
```

```
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



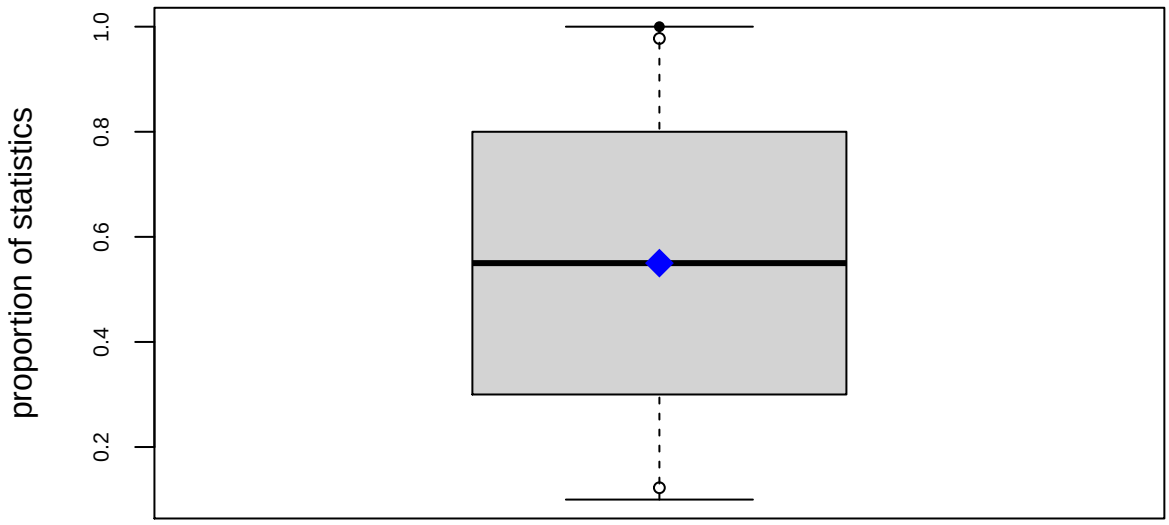

```
set.seed(12345)
mesamodel.02 <- ergm(mesa~edges)
```

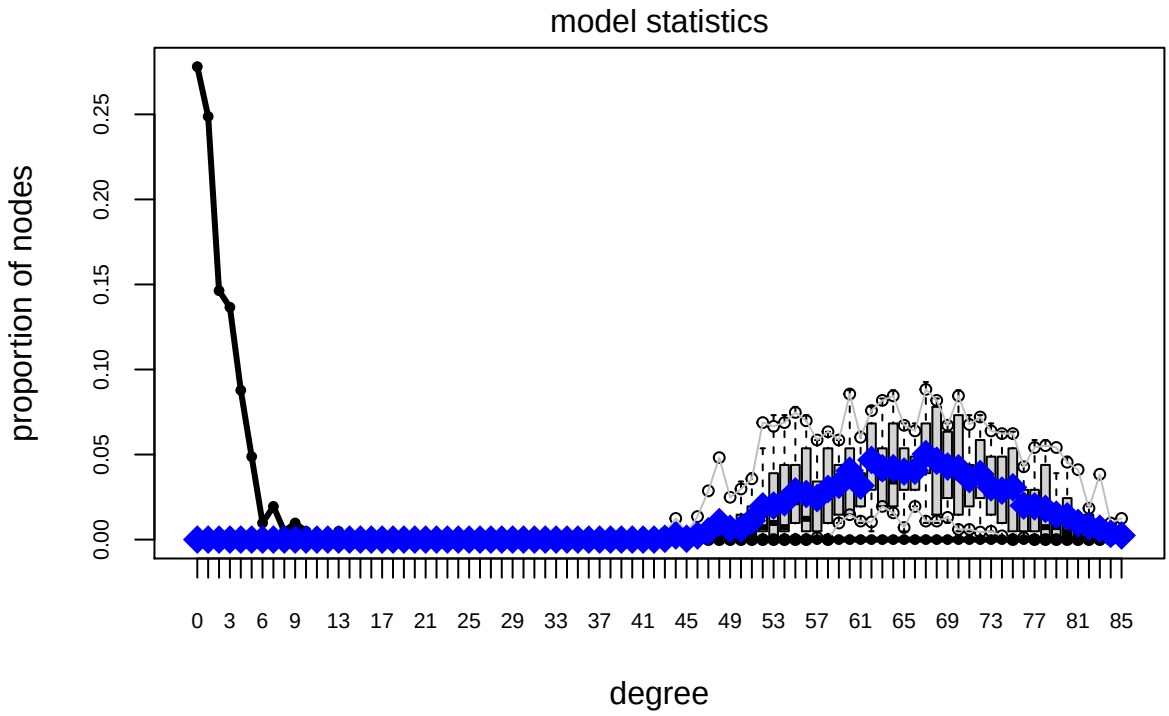
```
Starting maximum pseudolikelihood estimation (MPLE):

Evaluating the predictor and response matrix.

Maximizing the pseudolikelihood.

Finished MPLE.

Stopping at the initial estimate.

Evaluating log-likelihood at the estimate.
```

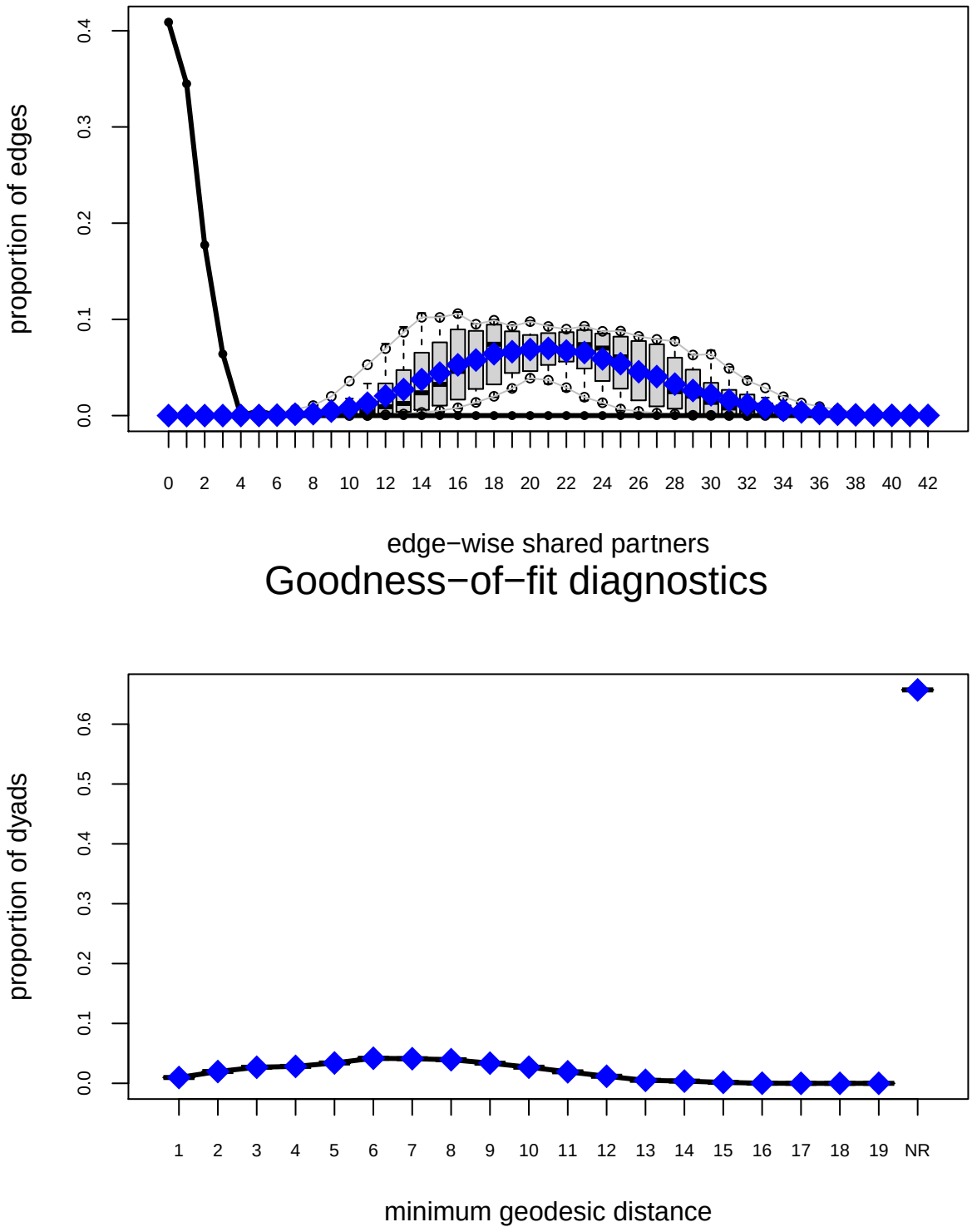
```
mesamodel.02.gof <- gof(mesamodel.02~degree + esp + distance,
                        control = snctrl(nsim=10))
```

```
Warning in gof.formula(object = object$formula, coef = coef, GOF = GOF, : No
parameter values given, using 0.
```

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


proportion of statistics
0.2
0.4
0.6
0.8
1.0


model statistics
proportion of nodes
0.00
0.05
0.10
0.15
0.20
0.25
0 3 6 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85
degree



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

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.

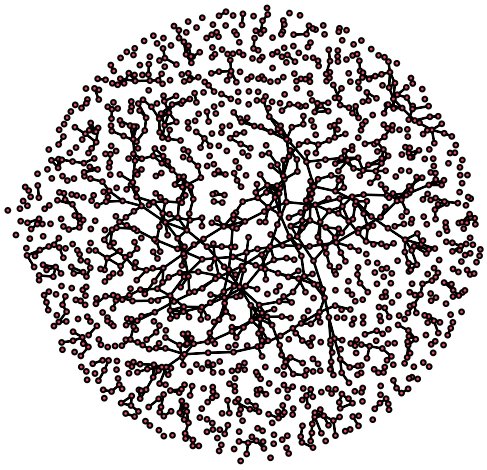
What it looks like when a model fails

For this, we'll using a larger network, faux.magnolia.high, and look at a simple model for triad closure.

```
set.seed(10)
data('faux.magnolia.high')
magnolia <- faux.magnolia.high
magnolia
```

```
 Network attributes:
  vertices = 1461
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 974
    missing edges= 0
    non-missing edges= 974

 Vertex attribute names:
    Grade Race Sex vertex.names

 Edge attribute names not shown
```

```
plot(magnolia, vertex.cex=.5)
```



```
# Consider a simple model
summary(magnolia~edges+triangle)
```

```
   edges triangle
     974      169
```

If we try to fit this "simple" model:

```
set.seed(100)
fit <- ergm(magnolia~edges+triangle,
            control=snctrl(MCMLE.effectiveSize=NULL))
```

```
Starting maximum pseudolikelihood estimation (MPLE):
Evaluating the predictor and response matrix.
Maximizing the pseudolikelihood.
Finished MPLE.
Starting Monte Carlo maximum likelihood estimation (MCMLE):
...
Iteration 4 of at most 60:
Optimizing with step length 0.3963.
The log-likelihood improved by 1.1568.
Estimating equations are not within tolerance region.
Iteration 5 of at most 60:
Error in ergm.MCMLE(init, nw, model, initialfit = (initialfit <- NULL),  :
  Number of edges in a simulated network exceeds that in the observed by a factor of more than 20. This
```

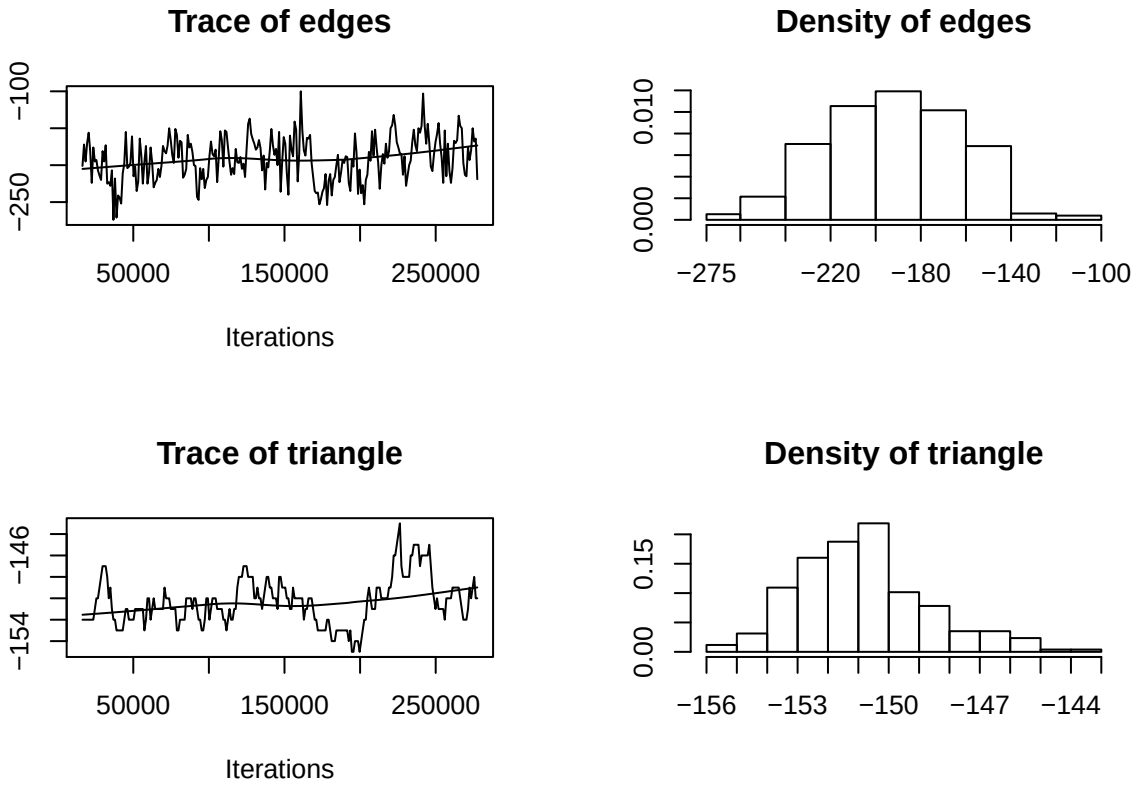
Very interesting. Instead of converging, the algorithm heads off into networks that are much much more dense than the observed network. This is such a clear indicator of a degenerate model specification that the algorithm stops after 3 iterations, to avoid heading off into areas that would cause memory 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:

```
set.seed(1000)
fit <- ergm(magnolia~edges+triangle,
            control=snctrl(MCMLE.maxit=2,MCMLE.effectiveSize=NULL))
```

```
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 2:
Optimizing with step length 0.2805.
```

```
The log-likelihood improved by 3.0798.
Estimating equations are not within tolerance region.
Iteration 2 of at most 2:
Optimizing with step length 0.0420.
The log-likelihood improved by 4.6627.
Estimating equations are not within tolerance region.
MCMLE estimation did not converge after 2 iterations. The estimated coefficients may not be accurate. Es
Finished MCMLE.
Evaluating log-likelihood at the estimate. Fitting the dyad-independent submodel...
Bridging between the dyad-independent submodel and the full model...
Setting up bridge sampling...
Using 16 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 .
Bridging finished.
This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the mcmc.diag
```

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



Clearly, this Markov chain is heading somewhere very bad.

This turns out to be one of the classic degenerate model specifications, and we now understand much more about why it does not produce reasonable levels of triadic closure.

We also now have a more robust way 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)

Let's see what using `gwesp` instead of `triangle` can do. We can also control the number of Metropolis-Hastings (MCMC) proposals between sampled statistics in our Markov chain, one of the many control parameters that may be passed to functions in the `ergm` package using the `control=snctrl()' syntax (to see the many control parameters that may be set by the user in the`ergm`package, type`?snctrl‘):

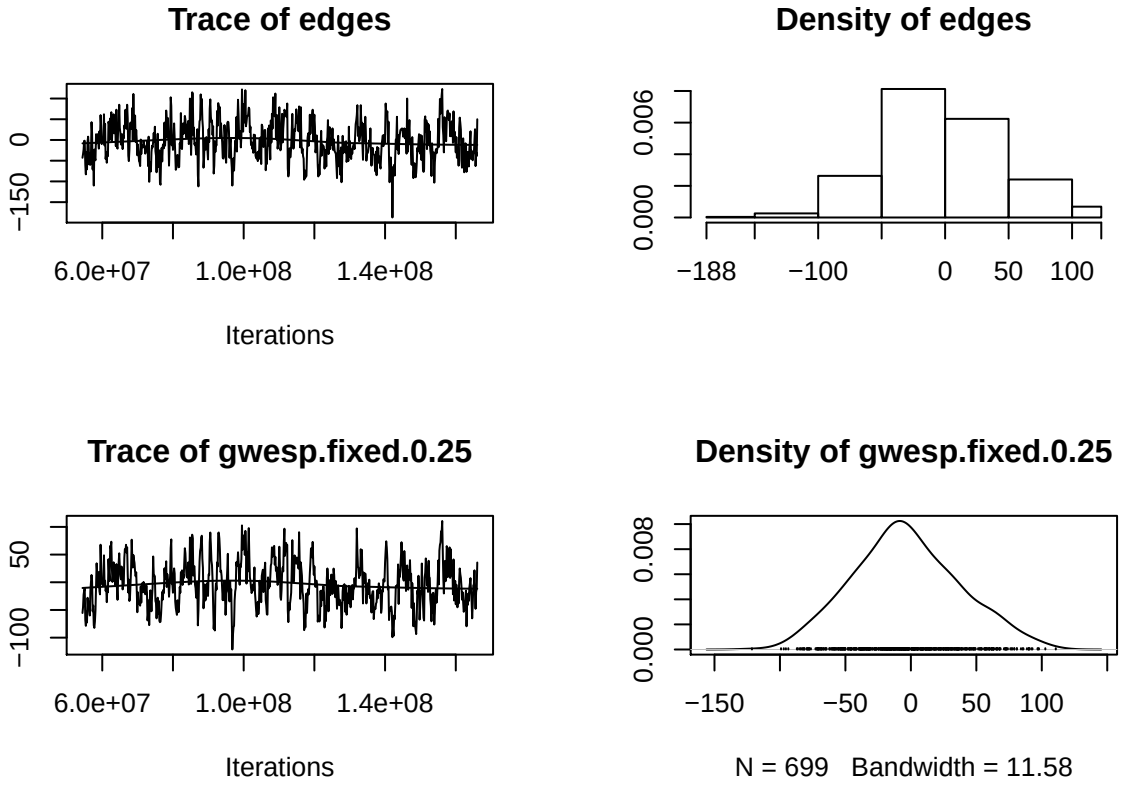
```
set.seed(10101)
fit <- ergm(magnolia~edges+gwesp(0.25, fixed=T),
            control=snctrl(MCMC.interval = 10000),
            verbose=T)
```

```
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):

 ... (output snipped)

Bridging finished.
This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the mcmc.dia
```

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

```
Sample statistics summary:

Iterations = 54400000:166080000
Thinning interval = 160000
Number of chains = 1
Sample size per chain = 699

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

                   Mean    SD Naive SE Time-series SE
edges            -2.342 46.56    1.761          4.311
gwesp.fixed.0.25 -3.257 40.49    1.531          4.088

2. Quantiles for each variable:

                   2.5%    25%    50%   75% 97.5%
edges            -87.00 -33.00 -5.000 29.00 88.55
gwesp.fixed.0.25 -78.78 -32.28 -5.507 24.33 79.00


Are sample statistics significantly different from observed?
                edges gwesp.fixed.0.25 Overall (Chi^2)
diff.      -2.3419170       -3.2570778              NA
test stat. -0.5432043       -0.7966579        1.752129
P-val.      0.5869891        0.4256497        0.418956

Sample statistics cross-correlations:
                     edges gwesp.fixed.0.25
edges            1.0000000        0.8582994
```

```
gwesp.fixed.0.25 0.8582994        1.0000000

Sample statistics auto-correlation:
Chain 1
               edges gwesp.fixed.0.25
Lag 0      1.0000000        1.0000000
Lag 160000 0.5688182        0.7267173
Lag 320000 0.4404817        0.5558670
Lag 480000 0.3531841        0.4300124
Lag 640000 0.2739027        0.3177659
Lag 8e+05  0.2258602        0.2534575

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
         0.50213          0.09449

Individual P-values (lower = worse):
           edges gwesp.fixed.0.25
       0.6155740        0.9247214
Joint P-value (lower = worse):  0.4598951 .
```




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

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.

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.

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

Long regarded as the poor country cousin in the network data family, egocentric data contain a remarkable amount of information. With the right statistical methods, such data can be used to explore the properties of the complete networks in which they are embedded. The basic idea here is to combine what is observed, with assumptions, to define a class of models that describe the distribution of networks that are centered on the observed properties. The variation in these networks quantifies some of the uncertainty introduced by the assumptions.

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

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

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

9. Additional functionality in statnet and other packages

`statnet` is actually 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, analysis of egocentrically sampled network data, 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.

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

Many of these have associated training workshops. Our tutorials can be found online, on the GitHub statnet Workshops wiki.

Additional functionality in base `ergm`

- ERGMs for valued ties

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

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
Samuel M. Jenness samuel.m.jenness@emory.edu

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

For a general orientation to the `statnet` packages, the best place to start is the special volume of the *Journal of Statistical Software* (JSS) devoted to `statnet`: https://www.jstatsoft.org/issue/view/v024. The nine papers in this volume cover a wide range of theoretical and practical topics related to ERGMs, and their implementation in `statnet`.

However, this volume was written in 2008. The `statnet` code base has evolved considerably since that time, and with the release of `ergm` version 4.0, the most current paper describing the capabilities of the `ergm` package is the following preprint:

Krivitsky, P. N., David R. Hunter, Martina Morris, and Chad Klumb (2021). ergm 4.0: New Features and Improvements. https://arxiv.org/abs/2106.04997.

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