ERGM LAB: Social Networks and Health Workshop 2017

Zack W Almquist (University of Minnesota)

May 15, 2017

Contents

Preliminaries

Basis of Workshop and History of its Development

This tutorial is based on the Sunbelt 2016 version produced by the Statnet Development Team:

- Martina Morris (University of Washington)
- Mark S. Handcock (University of California, Los Angeles)
- Carter T. Butts (University of California, Irvine)

- David R. Hunter (Penn State University)
- Steven M. Goodreau (University of Washington)
- Skye Bender de-Moll (Oakland)
- Pavel N. Krivitsky (University of Wollongong)

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

Modifications have been made by Zack Almquist (University of Minnesota) for the Social Networks and Health Workshop at Duke University May 25th, 2017.

Installation and Getting Started

Open an R session (in this lab we assume that you are using RStudio and GitHub). We also recommend you install Latex (windows, OSX). Installing latex will allow you to compile pdf RMarkdown documents for reproducible lab reports.

knitr

To use fully appreciate this lab you will need the R package knitr:

```
R> install.packages("knitr")
```

statnet

We will not be using all of statnet so you can choose to install: sna, network, ergm and coda only or you can install the full statnet package (commented out with #).

```
R> install.packages('sna')
R> install.packages('network')
R> install.packages('ergm')
R> install.packages('coda)
R> #install.packages('statnet')
```

If you have the packages already installed, it is recommended that you update them:

```
R> update.packages("name.of.package")
```

Loading the packages

```
R> library(ergm)
R> library(sna)
R> library(coda)
```

Check package version

```
R> sessionInfo()
```

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

```
R> set.seed(0)
```

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

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

The general form for an ERGM can be written as:

$$\Pr(Y = y) = \frac{\exp(\theta^T g(y, X))}{\kappa(y, \mathcal{Y}, X)}$$

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

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

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

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

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

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

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

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

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

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

```
R> data(package = "ergm")  # tells us the datasets in our packages
```

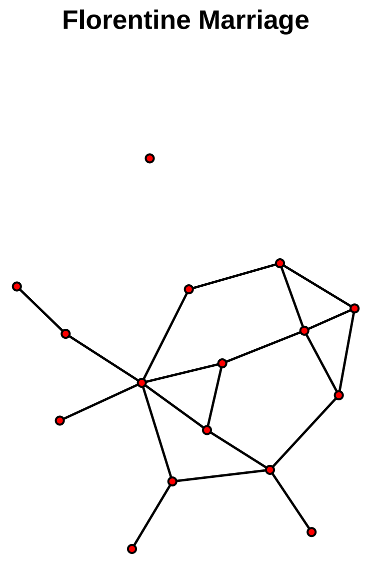
florentine Data

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

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

 Vertex attribute names:
    priorates totalties vertex.names wealth

No edge attributes
```

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

```
edges
   20
```



Bernoulli model

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

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

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

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

```

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

Formula:   flomarriage ~ edges

Iterations:  5 out of 20

Monte Carlo MLE Results:
      Estimate Std. Error MCMC % p-value
edges  -1.6094     0.2449      0  <1e-04 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 166.4  on 120  degrees of freedom
 Residual Deviance: 108.1  on 119  degrees of freedom

AIC: 110.1    BIC: 112.9    (Smaller is better.)
```

How should we interpret the coefficient from this model? The log-odds of any tie existing is:

$$= -1.6094379 \times \text{change in the number of ties}$$
$$= -1.6094379 \times 1$$

Triad formation

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

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

```
   edges triangle
      20        3
```

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

```
Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 0.005728
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by < 0.0001
Step length converged twice. Stopping.
Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
```

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

```r
R> summary(flomodel.02)
```

```

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

Formula:   flomarriage ~ edges + triangle

Iterations:  2 out of 20

Monte Carlo MLE Results:
         Estimate Std. Error MCMC % p-value
edges     -1.6694     0.3521      0  <1e-04 ***
triangle   0.1532     0.5771      0   0.791
---
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.)
```

Now, how should we interpret coefficients?

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

$$-1.6693954 \times \text{change in the number of ties} + 0.153229 \times \text{change in the number of triangles}$$

- For a tie that will create no triangles, the conditional log-odds is: -1.6693954
- if one triangle: -1.6693954 + 0.153229 = -1.5161664
- if two trianlgles: -1.6693954 + 0.153229 $\times$2 = -1.3629374
- the corresponding probabilities are 0.16, 0.18, 0.2.

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

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

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

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

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

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

```
    edges  triangle
-1.669395  0.153229
```

Nodal covariates: effects on mean degree

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

```
R> wealth <- flomarriage %v% "wealth"  # %v% references vertex attributes
R> wealth
```

```
 [1]  10  36  55  44  20  32   8  42 103  48  49   3  27  10
[15] 146  48
```

```
R> 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
```

```
R> plot(flomarriage, vertex.cex = wealth/25, main = "Florentine marriage by wealth",
+      cex.main = 0.8)  # network plot with vertex size proportional to wealth
R> summary(flomarriage ~ edges + nodecov("wealth"))
```

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

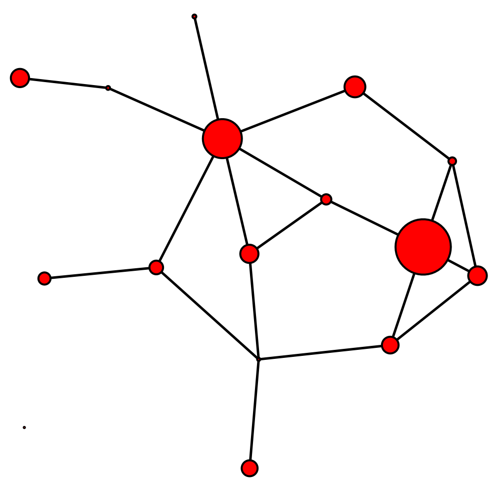
```
R> # observed statistics for the model
R> flomodel.03 <- ergm(flomarriage ~ edges + nodecov("wealth"))
```

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

```
R> summary(flomodel.03)
```

```

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

Formula:   flomarriage ~ edges + nodecov("wealth")

Iterations:  4 out of 20

Monte Carlo MLE Results:
                Estimate Std. Error MCMC % p-value
edges          -2.594929   0.536056      0  <1e-04 ***
nodecov.wealth  0.010546   0.004674      0  0.0259 *
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 166.4  on 120  degrees of freedom
 Residual Deviance: 103.1  on 118  degrees of freedom

AIC: 107.1    BIC: 112.7    (Smaller is better.)
```

Florentine marriage by wealth



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

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

$$-2.594929 \times \text{change in the number of ties} + 0.0105459 \times \text{the wealth of node 1} + 0.0105459 \times \text{the wealth of node 2}$$

$$-2.594929 \times \text{change in the number of ties} + 0.0105459 \times \text{the sum of the wealth of the two nodes}$$

- for a tie between two nodes with minimum wealth, the conditional log-odds is:
  - -2.594929 + 0.0105459 $\times(3+3)$ = -2.5316536
- for a tie between two nodes with maximum wealth:
  - -2.594929 + 0.0105459 $\times(146+146)$ = 0.4844775
- for a tie between the node with maximum wealth and the node with minimum wealth:
  - -2.594929 + 0.0105459 $\times(146+3)$ = -1.023588
- The corresponding probabilities are 0.07, 0.62, 0.26

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

Faux Mesa High

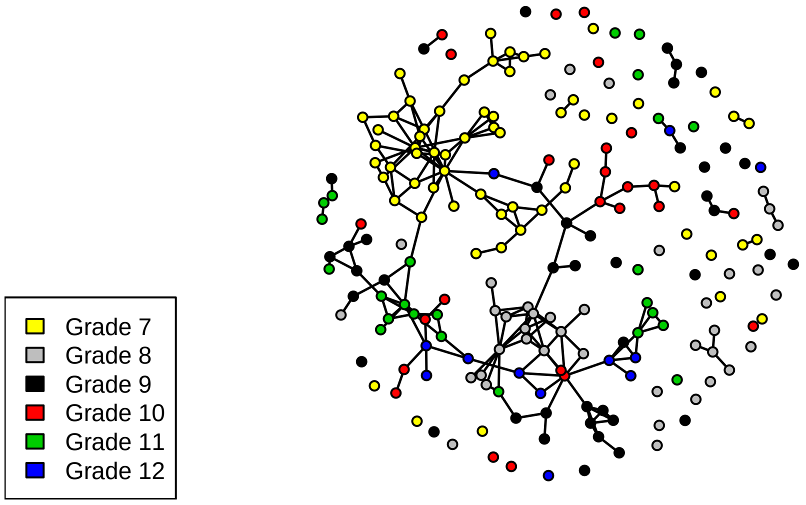
Let's try a larger network, a simulated mutual friendship network based on one of the schools from the Add Health study.

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

```
R> 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
```

```
R> par(mfrow = c(1, 1))  # Back to 1-panel plots
R> plot(mesa, vertex.col = "Grade")
R> legend("bottomleft", fill = 7:12, legend = paste("Grade", 7:12),
+     cex = 0.75)
```



Here, we'll examine the homophily in friendships by grade and race. Both are discrete attributes so we use the ergm-term nodematch.

```
R> fauxmodel.01 <- ergm(mesa ~ edges + nodematch("Grade", diff = T) +
+     nodematch("Race", diff = T))
```

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

```
R> summary(fauxmodel.01)
```

```


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

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

Iterations:  8 out of 20

Monte Carlo MLE Results:
                     Estimate Std. Error MCMC % p-value
```

```
edges                   -6.2328     0.1742      0  <1e-04 ***
nodematch.Grade.7        2.8740     0.1981      0  <1e-04 ***
nodematch.Grade.8        2.8788     0.2391      0  <1e-04 ***
nodematch.Grade.9        2.4642     0.2647      0  <1e-04 ***
nodematch.Grade.10       2.5692     0.3770      0  <1e-04 ***
nodematch.Grade.11       3.2921     0.2978      0  <1e-04 ***
nodematch.Grade.12       3.8376     0.4592      0  <1e-04 ***
nodematch.Race.Black       -Inf     0.0000      0  <1e-04 ***
nodematch.Race.Hisp      0.0679     0.1737      0  0.6959
nodematch.Race.NatAm     0.9817     0.1842      0  <1e-04 ***
nodematch.Race.Other       -Inf     0.0000      0  <1e-04 ***
nodematch.Race.White     1.2685     0.5371      0  0.0182 *
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

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

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

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

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

```
R> table(mesa %v% "Race")  # Frequencies of race
```

```
Black  Hisp NatAm Other White
    6   109    68     4    18
```

```
R> mixingmatrix(mesa, "Race")
```

```
Note:  Marginal totals can be misleading
 for undirected mixing matrices.
      Black Hisp NatAm Other White
Black     0    8    13     0     5
Hisp      8   53    41     1    22
NatAm    13   41    46     0    10
Other     0    1     0     0     0
White     5   22    10     0     4
```

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

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

```
R> summary(mesa ~ edges + nodematch("Grade", diff = T) + nodematch("Race",
+     diff = T))
```

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

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

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

Sampson Monk Data

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.

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

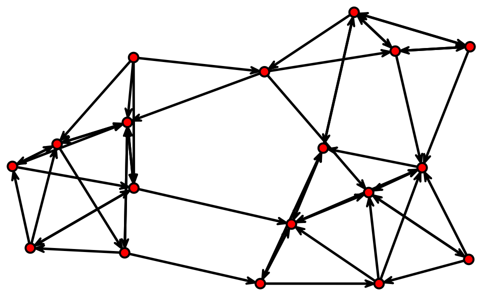
```
 [1] "faux.mesa.high" "fauxmodel.01"   "flobusiness"   
 [4] "flomarriage"    "flomodel.01"    "flomodel.02"   
 [7] "flomodel.03"    "ilogit"         "logit"         
[10] "mesa"           "samplk1"        "samplk2"       
[13] "samplk3"        "wealth"        
```

```
R> 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
```

```
R> plot(samplk3)
R> summary(samplk3 ~ edges + mutual)
```

```
 edges mutual 
    56     15 
```



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

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

Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by < 0.0001
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 0.0002585
Step length converged twice. Stopping.
Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 

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

```
R> summary(sampmodel.01)


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

Formula:   samplk3 ~ edges + mutual

Iterations:  2 out of 20

Monte Carlo MLE Results:
       Estimate Std. Error MCMC % p-value
edges   -2.1508     0.2130      0  <1e-04 ***
mutual   2.2936     0.4731      0  <1e-04 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 424.2  on 306  degrees of freedom
 Residual Deviance: 267.9  on 304  degrees of freedom

AIC: 271.9    BIC: 279.4    (Smaller is better.)
```

There is a strong and significant mutuality effect. The coefficients for the edges and mutual terms roughly cancel for a mutual tie, so the conditional odds of a mutual tie are about even, and the probability is about 0.5356413 %. By contrast a non-mutual tie has a conditional log-odds of 0.1042528, or 0.1042528 % probability.

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

Missing Data Example

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

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

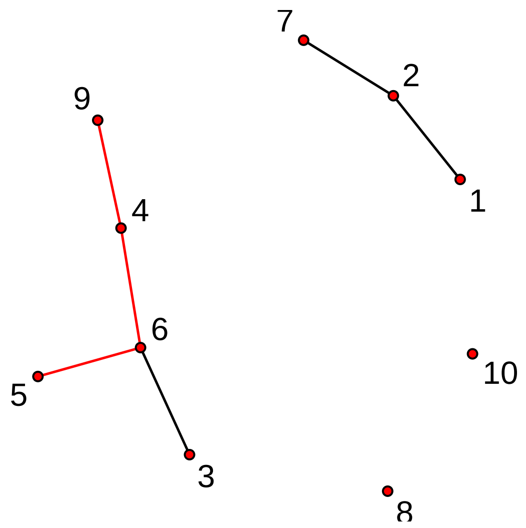
```
R> missnet <- network.initialize(10, directed = F)
R> missnet[1, 2] <- missnet[2, 7] <- missnet[3, 6] <- 1
R> missnet[4, 6] <- missnet[4, 9] <- missnet[5, 6] <- NA
R> summary(missnet)
```

```
Network attributes:
  vertices = 10
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
 total edges = 6
   missing edges = 3
   non-missing edges = 3
 density = 0.06666667

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

No edge attributes

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

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

```
edges
```

```
    3
R> summary(missnet.01 <- ergm(missnet ~ edges))

Evaluating log-likelihood at the estimate.


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

Formula:   missnet ~ edges

Iterations:  5 out of 20

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

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

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



```
R> missnet_bad <- missnet
R> missnet_bad[4, 6] <- missnet_bad[4, 9] <- missnet_bad[5, 6] <- 0
R> 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
```

```
R> summary(missnet.02 <- ergm(missnet_bad ~ edges))
```

```
Evaluating log-likelihood at the estimate.


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

Formula:   missnet_bad ~ edges

Iterations:  5 out of 20 

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

     Null Deviance: 62.38  on 45  degrees of freedom
 Residual Deviance: 22.04  on 44  degrees of freedom
 
AIC: 24.04    BIC: 25.85    (Smaller is better.) 
```

The coefficient is smaller now because the missing ties are counted as “0”, and translates to a conditional tie probability of 0.07%. 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.

ERGM TERMS

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

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

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

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

Terms provided with ergm

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

```
R> help("ergm-terms")
```

A table of commonly used terms can be found here.

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

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

Coding new ergm-terms

statnet has recently released a new package (*ergm.userterms*) that makes it much easier to write one's own ergm-terms. The package is available on CRAN, and installing it will include the tutorial (ergmuserterms.pdf). Alternatively, the tutorial can be found in the Journal of Statistical Software 52(2).

Note that writing up new ergm terms requires some knowledge of C and the ability to build R from source (although the latter is covered in the tutorial, the many environments for building R and the rapid changes in these environments make these instructions obsolete quickly).

Goodness of Fit

Network simulation: the simulate command and network.list objects

Once we have estimated the coefficients of an ERGM, the model is completely specified. It defines a probability distribution across all networks of this size. If the model is a good fit to the observed data, then networks drawn from this distribution will be more likely to "resemble"" the observed data. To see examples of networks drawn from this distribution we use the simulate command:

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

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

```
R> summary(flomodel.03.sim)
```

```
Number of Networks: 10
Model: flomarriage ~ edges + nodecov("wealth")
Reference: ~Bernoulli
Constraints: ~.
Parameters:
         edges nodecov.wealth
   -2.59492903     0.01054591

Stored network statistics:
      edges nodecov.wealth
 [1,]    22           1787
 [2,]    23           2742
 [3,]    21           2115
 [4,]    20           2163
 [5,]    25           2673
 [6,]    24           2943
 [7,]    24           2423
 [8,]    15           1501
 [9,]    19           2309
[10,]    20           2288
```

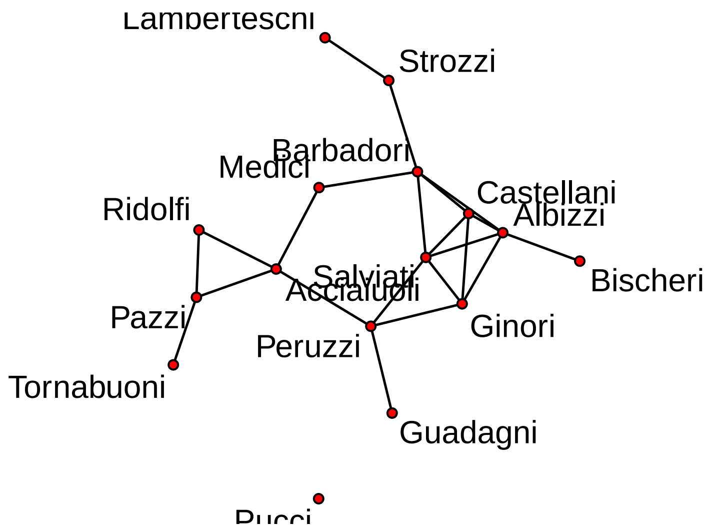
```
R> length(flomodel.03.sim)
```

```
[1] 10
```

```
R> 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
```

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



Voila. Of course, yours will look somewhat different.

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

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

Examining the quality of model fit - GOF

There are two types of goodness of fit (GOF) tests in ergm. The first, and one you'll always want to run, is used to evaluate how well the estimates are reproducing the terms that are in the model. These are maximum likelihood estimates, so they should reproduce the observed sufficient statistics well, and if they don't it's an indication that something may be wrong in the estimation process. Assuming all is well here, you can move on to the next step.

The second type of GOF test is used to see how well the model fits other emergent patterns in your network data, patterns that are not explicitly represented by the terms in the model. ERGMs are cross-sectional; they don't directly model the process of tie formation and dissolution (that would be a temporal ergm (TERGM), see the tergm package). But ERGMs can be seen as generative models in another sense: they represent the process that governs the emergent global pattern of ties from a local perspective.

To see this, it's worth digging a bit deeper into the MCMC estimation process. When you estimate an ERGM in statnet, the MCMC algorithm at each step draws a dyad at random, and evaluates the probability of a tie from the perspective of these two nodes. That probability is governed by the ergm-terms in the model, and the current estimates of the coefficients on these terms. Once the estimates converge, simulations from the model will produce networks that are centered on the observed model statistics (as we saw above), but the networks will also have other emergent global properties, even though these global properties are not represented by explicit terms in the model. If the local processes represented by the model terms capture the true underlying process, the model should reproduce these global properties as well.

So the second of whether a local model "fits the data" is to evaluate how well it reproduces observed global network properties that are not in the model. We do this by choosing network statistics that are not in the model, and comparing the value observed in the original network to the distribution of values we get in simulated networks from our model.

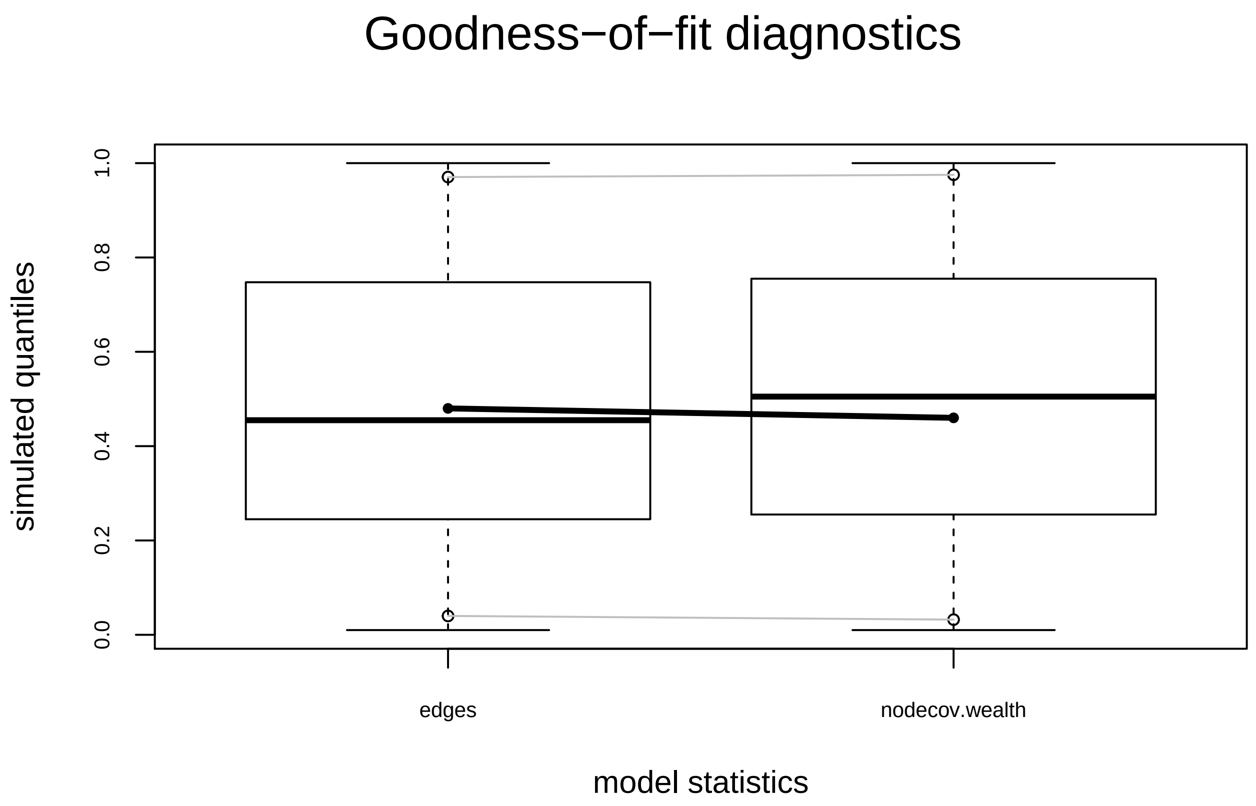
Both types of tests can be conducted by using the gof function. The gof function is a bit different than the summary, ergm, and simulate functions, in that it currently only takes 4 possible arguments: model, degree, esp (edgwise share partners), and distance (geodesic distances). “model” uses gof to evaluate the fit to the terms in the model, and the other 3 terms are used to evaluate the fit to emergent global network properties, at either the node level (degree), the edge level (esp), or the dyad level (distance). Note that these 3 global terms represent distributions, not single number summary statistics.

Let’s start by looking at how to assess the fit of the model to the terms in the model, using model 3 from the flomarriage example ().

```
R> flo.03.gof.model <- gof(flomodel.03 ~ model)
R> flo.03.gof.model
```

```

Goodness-of-fit for model statistics

                obs  min    mean  max MC p-value
edges            20   11   19.87   32       0.96
nodecov.wealth 2168 1180 2163.58 3356       0.92
```

```
R> plot(flo.03.gof.model)
```



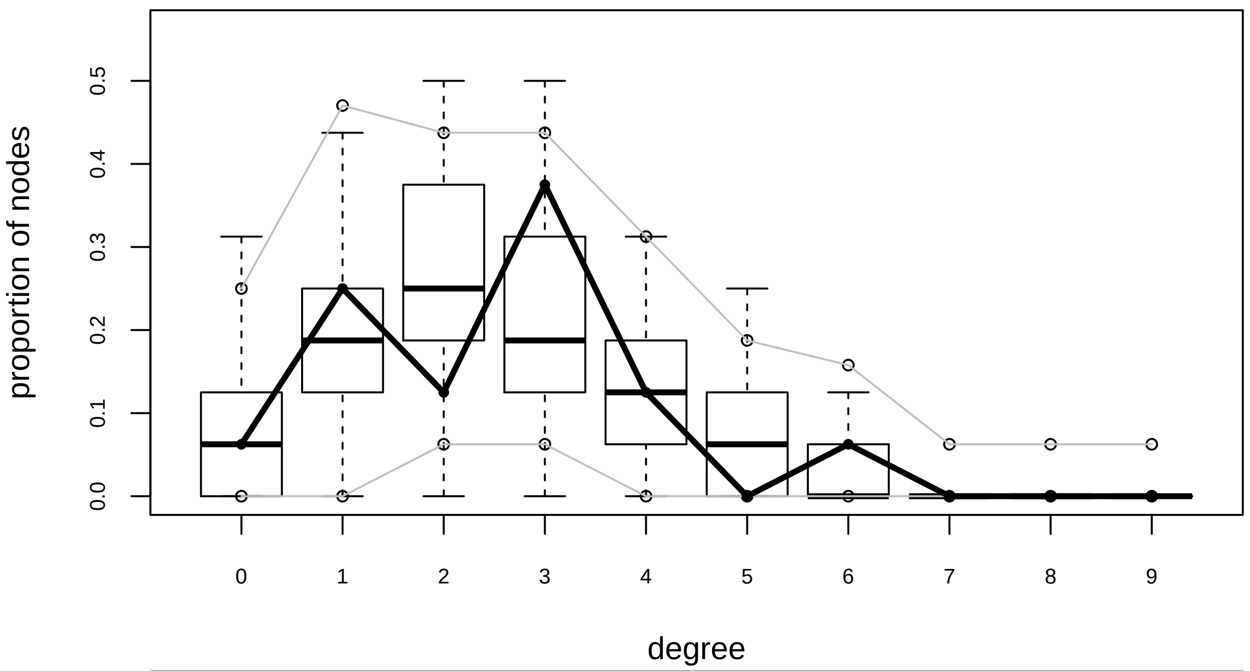
Looks pretty good. Now let’s look at the fit to the 3 global network terms that are not in the model.

```
R> flo.03.gof.global <- gof(flomodel.03 ~ degree + esp + distance)
R> flo.03.gof.global
```

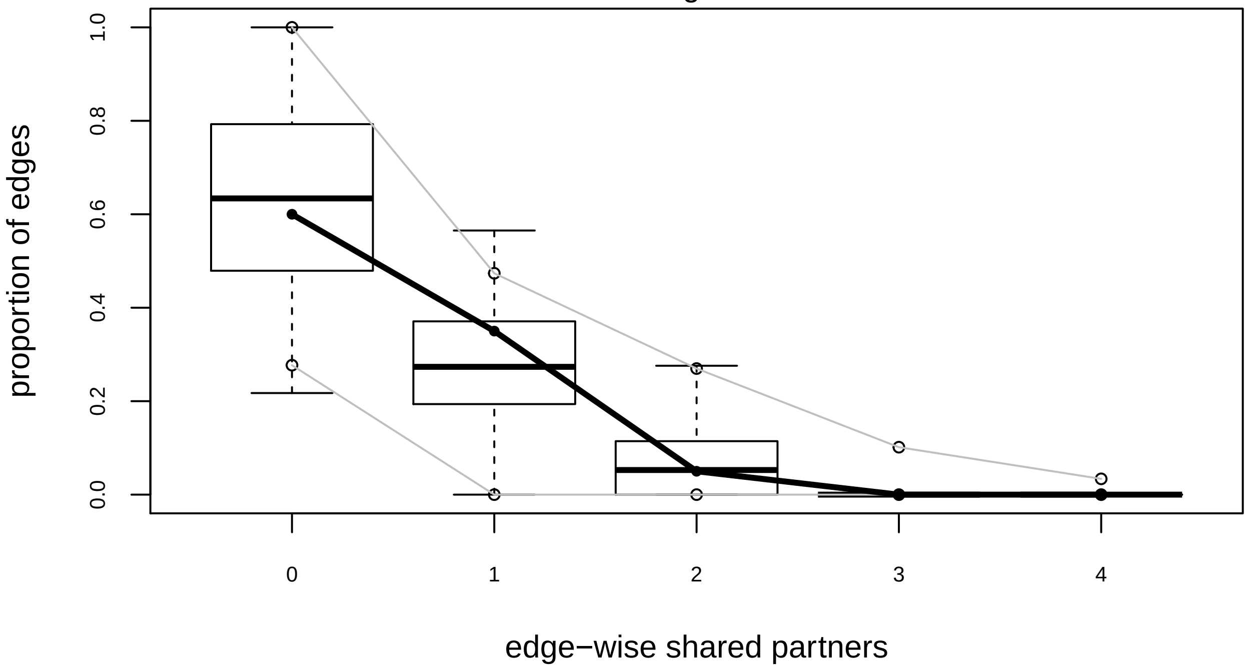
```

Goodness-of-fit for degree
```

```
   obs min mean max MC p-value
0    1   0 1.31   5       1.00
1    4   0 3.10   9       0.76
2    2   0 4.32   8       0.32
3    6   0 3.42   8       0.22
4    2   0 2.05   5       1.00
5    0   0 1.02   4       0.66
6    1   0 0.47   4       0.70
7    0   0 0.20   2       1.00
8    0   0 0.06   1       1.00
9    0   0 0.04   1       1.00
10   0   0 0.01   1       1.00

Goodness-of-fit for edgewise shared partner

     obs min  mean max MC p-value
esp0  12   5 12.45  20       1.00
esp1   7   0  5.69  13       0.76
esp2   1   0  1.71   9       1.00
esp3   0   0  0.32   7       1.00
esp4   0   0  0.05   2       1.00
esp5   0   0  0.01   1       1.00

Goodness-of-fit for minimum geodesic distance

    obs min  mean max MC p-value
1    20  13 20.23  35       1.00
2    35  12 34.06  58       0.98
3    32   8 26.41  42       0.54
4    15   0 12.08  27       0.80
5     3   0  3.80  16       0.98
6     0   0  1.05   9       1.00
7     0   0  0.19   4       1.00
8     0   0  0.03   2       1.00
Inf  15   0 22.15  74       1.00
```

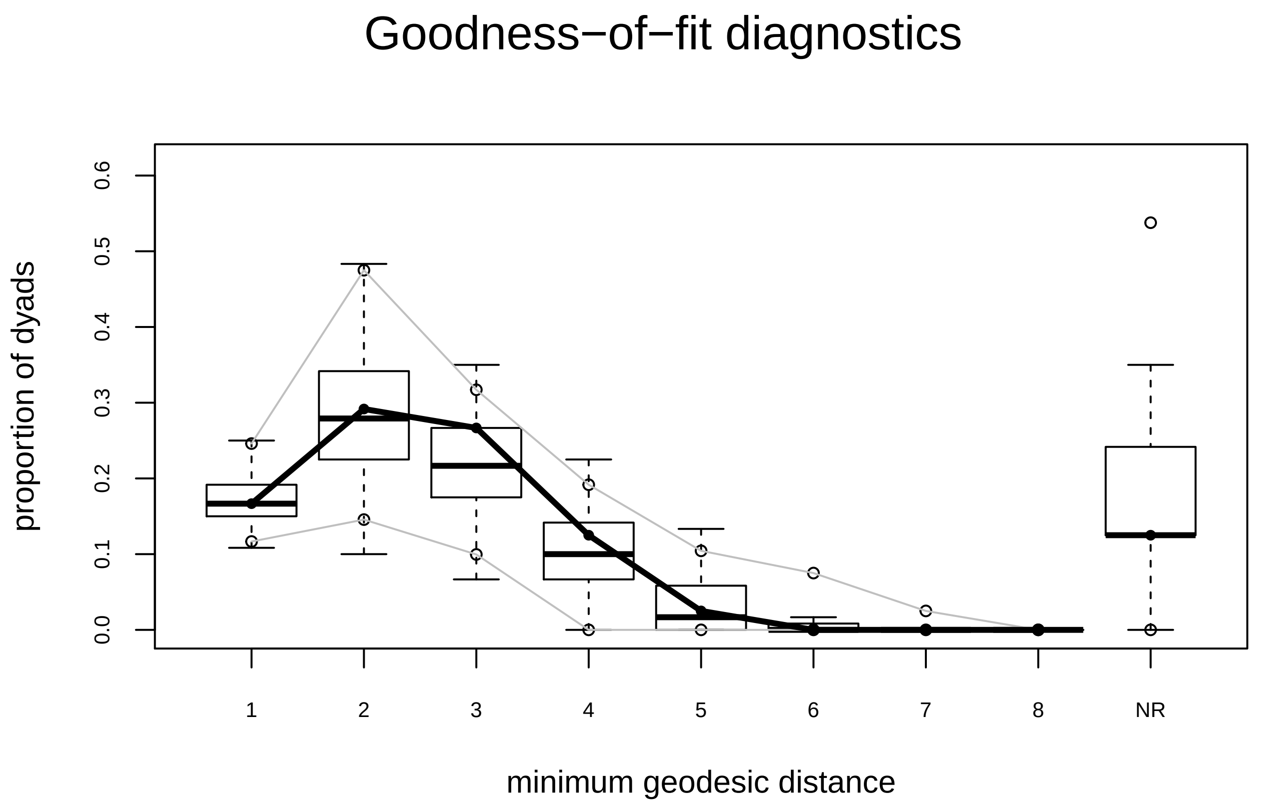
```
R> plot(flo.03.gof.global)
```



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



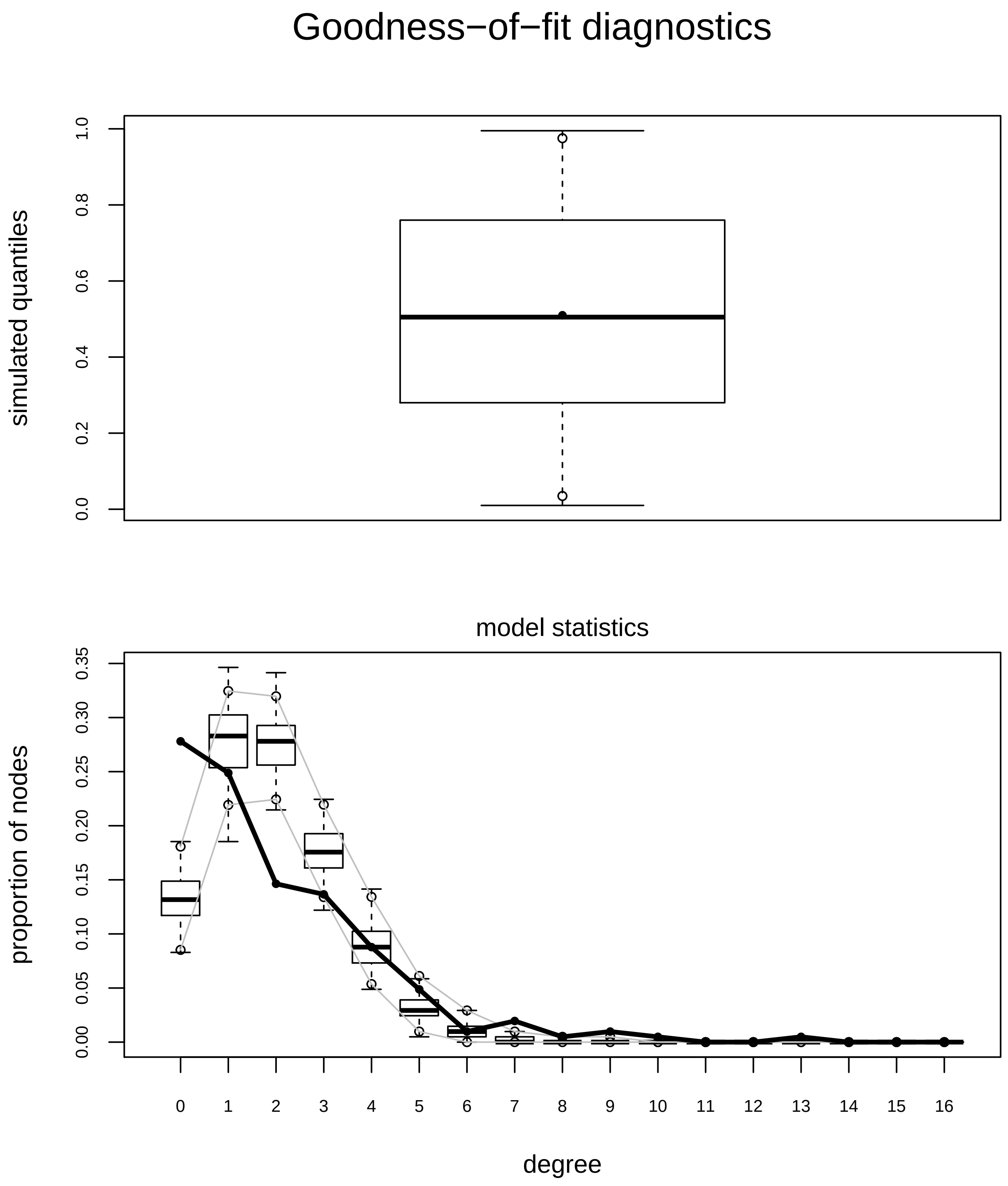
These look pretty good too.

Let's compare this to a simple Bernoulli model for faux.mesa.high.

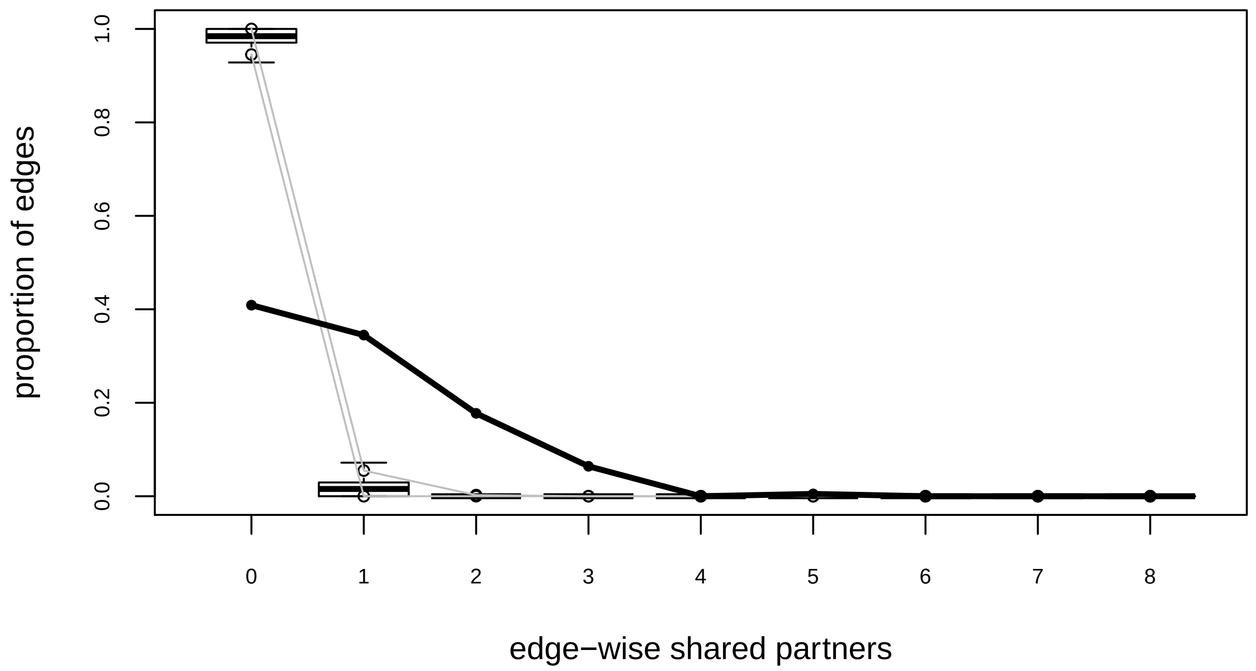
```
R> mesamodel.b <- ergm(mesa ~ edges)
```

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

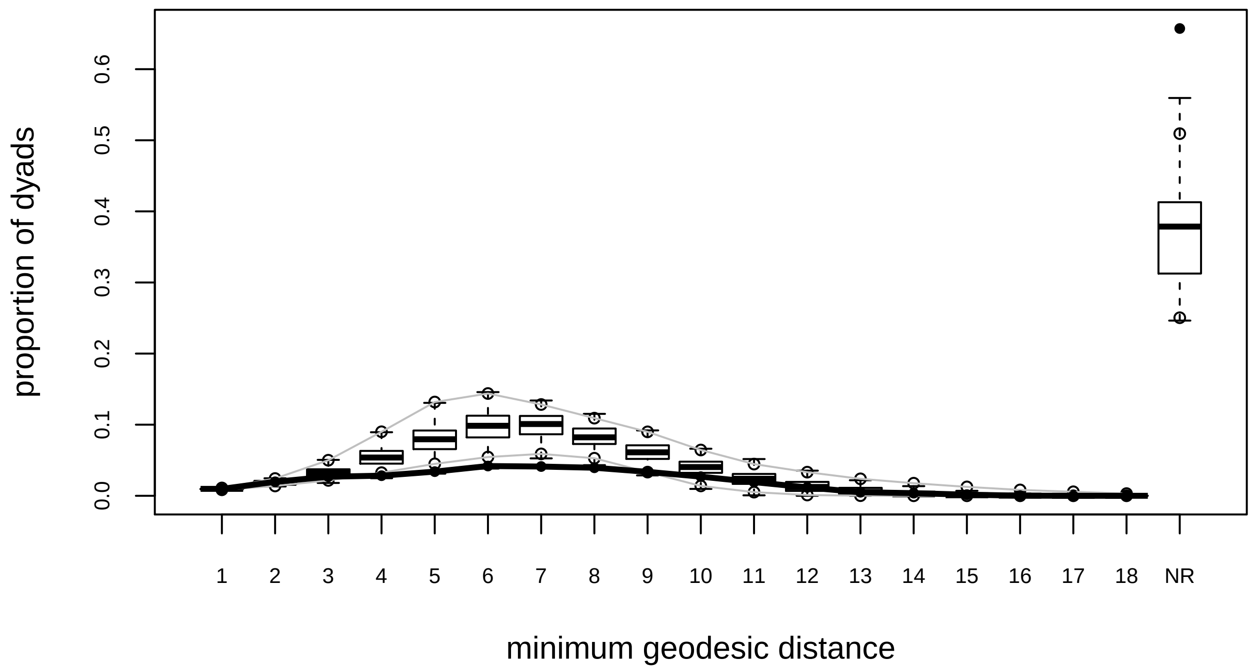
```
R> plot(gof(mesamodel.b ~ model, nsim = 10))
R> plot(gof(mesamodel.b ~ degree + esp + distance, nsim = 10))
```



Goodness−of−fit diagnostics
simulated quantiles
0.0
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
0.30
0.35
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
degree



Goodness−of−fit diagnostics



These plots suggest this is not a very good model – while it reproduces the edges term included in the model (which means all is well with the estimation) it does not do a good job of capturing the rest of the network structure.

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

MCMC Diagnostics

Diagnostics: troubleshooting and checking for model degeneracy

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

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

For more detailed discussion of model degeneracy in the ERGM context, see the papers by Mark Handcock referenced at the end of this tutorial.

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

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

What it looks like when a model converges properly

We will first consider a simulation where the algorithm works using the program defaults, and observe the behavior of the MCMC estimation algorithm using the mcmc.diagnostics function. This function allows youto evaluate how well the estimation algorithm is "mixing' – that is, how much serial correlation there is in the MCMC sample estimates – and whether it is converging around an estimate, or heading off in a trend up or down. The latter is an indication of poor (or no) model convergence.

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

```
  edges degree1 
     15       3 
```

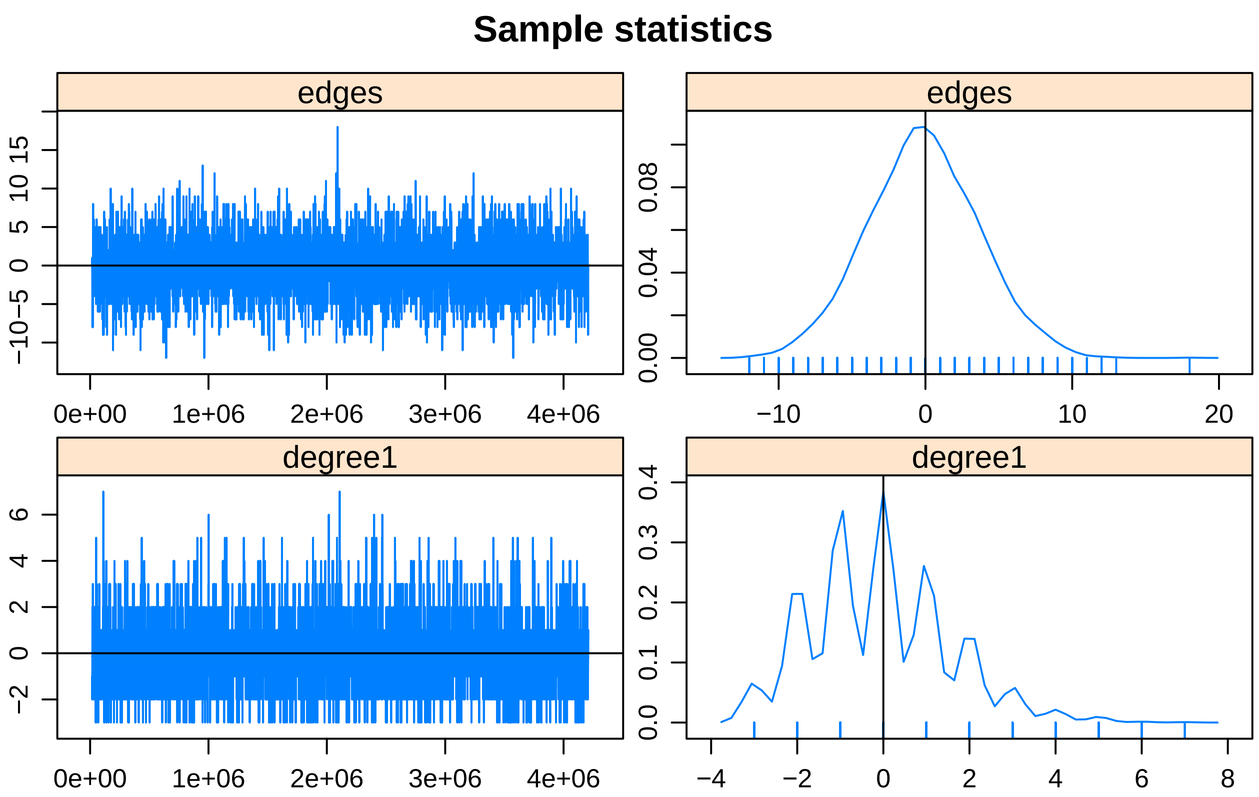
```
R> fit <- ergm(flobusiness ~ edges + degree(1))
```

```
Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 0.2726
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 0.003505
Step length converged twice. Stopping.
Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

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

```
R> mcmc.diagnostics(fit)
```

```
Sample statistics summary:

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

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

           Mean    SD Naive SE Time-series SE
edges   -0.13843 3.789  0.05920        0.05787
degree1 -0.08569 1.597  0.02496        0.02496

2. Quantiles for each variable:

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


Sample statistics cross-correlations:
             edges    degree1
edges    1.0000000 -0.4111134
degree1 -0.4111134  1.0000000

Sample statistics auto-correlation:
Chain 1
                edges      degree1
Lag 0     1.000000000  1.000000000
Lag 1024 -0.022737259  0.006308238
Lag 2048 -0.005012976  0.004218457
Lag 3072 -0.023239391  0.011116619
Lag 4096 -0.008293901 -0.015689309
Lag 5120 -0.017343019 -0.009461073

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

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

  edges degree1
-1.5712  0.5487

Individual P-values (lower = worse):
    edges   degree1
0.1161443 0.5831907
Joint P-value (lower = worse):  0.34696 .


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



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

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

```
R> summary(bad.model <- ergm(flobusiness ~ edges + degree(1), control = control.ergm(MCMC.interval = 1))
```

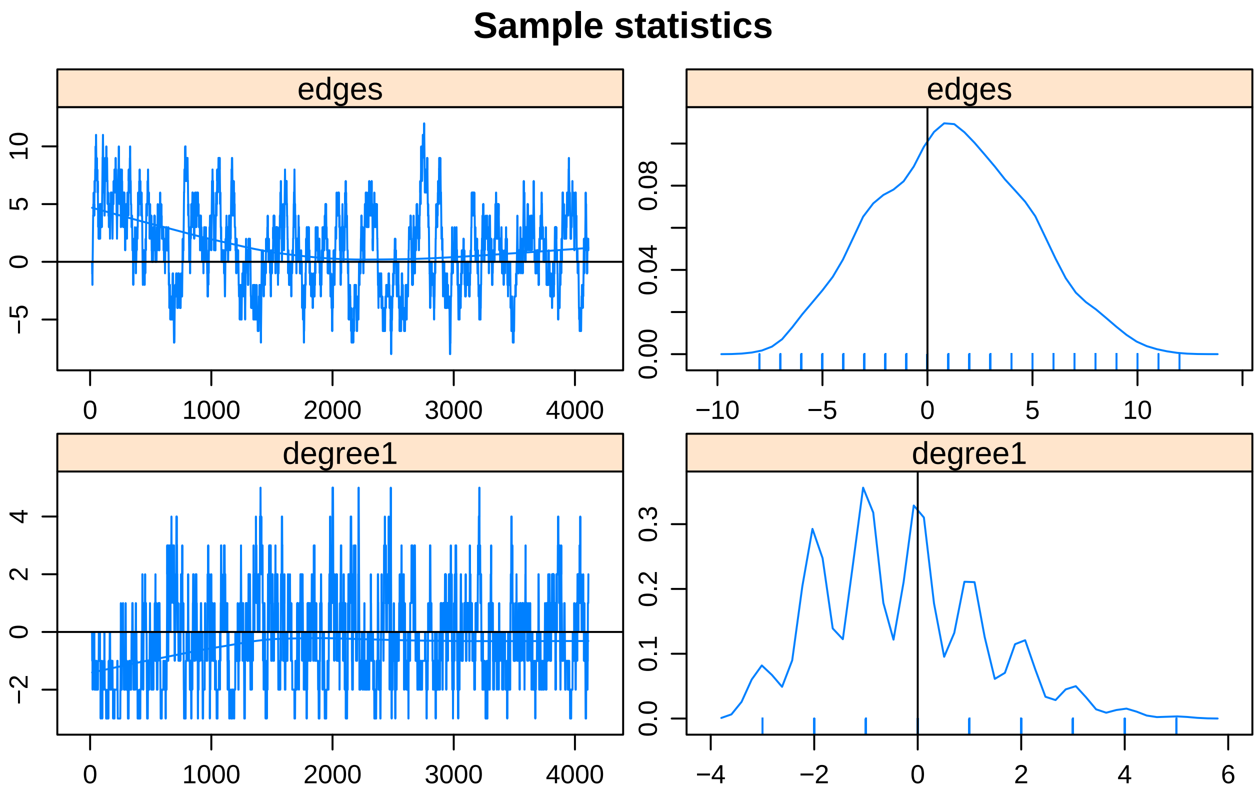
```
Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 0.1629
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 0.05638
Step length converged twice. Stopping.
Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 

This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the mcmc.diag


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

Formula:   flobusiness ~ edges + degree(1)
```

```
Iterations:  2 out of 20

Monte Carlo MLE Results:
        Estimate Std. Error MCMC % p-value
edges    -2.1309     0.3292      1  <1e-04 ***
degree1  -0.6907     0.7478      0   0.358
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 166.36  on 120  degrees of freedom
 Residual Deviance:  89.54  on 118  degrees of freedom

AIC: 93.54    BIC: 99.12    (Smaller is better.)
```

```
R> mcmc.diagnostics(bad.model)
```

```
Sample statistics summary:

Iterations = 16:4111
Thinning interval = 1
Number of chains = 1
Sample size per chain = 4096

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

           Mean    SD Naive SE Time-series SE
edges    1.1799 3.550  0.05547         0.4550
degree1 -0.3269 1.556  0.02431         0.1158

2. Quantiles for each variable:

        2.5% 25% 50% 75% 97.5%
edges     -5  -1   1   4     8
degree1   -3  -2   0   1     3


Sample statistics cross-correlations:
            edges   degree1
edges    1.000000 -0.508268
degree1 -0.508268  1.000000

Sample statistics auto-correlation:
Chain 1
          edges   degree1
Lag 0 1.0000000 1.0000000
Lag 1 0.9706983 0.9003196
Lag 2 0.9431987 0.8119068
Lag 3 0.9170167 0.7291452
Lag 4 0.8919744 0.6563743
Lag 5 0.8679203 0.5977316

Sample statistics burn-in diagnostic (Geweke):
```

```
Chain 1

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

  edges degree1
  4.934  -6.219

Individual P-values (lower = worse):
       edges      degree1
8.038557e-07 4.999380e-10
Joint P-value (lower = worse):  0.002432263 .


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



This runs a version with every network returned, and might be useful if you are trying to debug a bad model fit. (We'll leave you to explore this object more on your own)

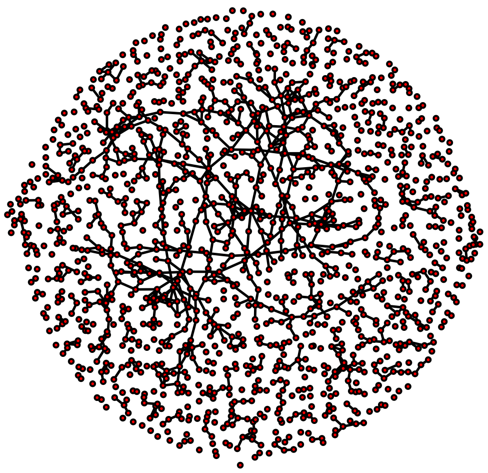
What it looks like when a model fails

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

```
R> data("faux.magnolia.high")
R> magnolia <- faux.magnolia.high
R> plot(magnolia, vertex.cex = 0.5)
R> summary(magnolia ~ edges + triangle)
```

```
   edges triangle
```

```
     974      169
```

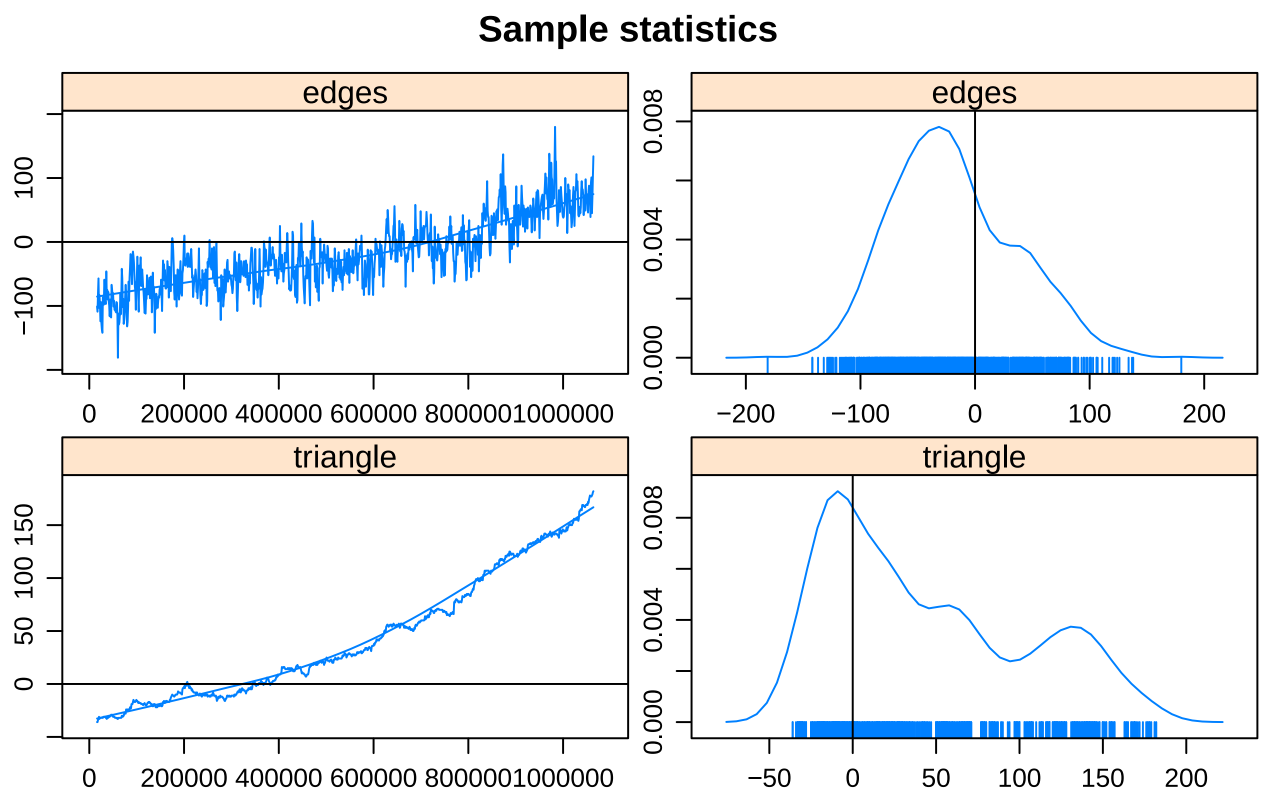


Very interesting. In the process of trying to fit this model, the algorithm heads off into networks that are much much more dense than the observed network. This is such a clear indicator of a degenerate model specification that the algorithm stops after 3 iterations, to avoid heading off into areas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

```
R> fit.mag.01 <- ergm(magnolia ~ edges + triangle, control = control.ergm(MCMLE.maxit = 2))

Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 2:
The log-likelihood improved by 3.838
Iteration 2 of at most 2:
The log-likelihood improved by 2.114
Step length converged once. Increasing MCMC sample size.
Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

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

```
R> mcmc.diagnostics(fit.mag.01)

Sample statistics summary:

Iterations = 16384:1063936
Thinning interval = 1024
Number of chains = 1
Sample size per chain = 1024

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

           Mean    SD Naive SE Time-series SE
edges    -18.76 53.44    1.670          18.05
triangle  43.88 58.97    1.843          42.10

2. Quantiles for each variable:

         2.5% 25%   50% 75% 97.5%
edges    -109 -56 -24.5  16  90.0
triangle  -32  -8  27.0  85 164.4
```

```
Sample statistics cross-correlations:
           edges triangle
edges    1.00000  0.85678
triangle 0.85678  1.00000

Sample statistics auto-correlation:
Chain 1
             edges  triangle
Lag 0    1.0000000 1.0000000
Lag 1024 0.8872624 0.9961728
Lag 2048 0.8256264 0.9924101
Lag 3072 0.7848450 0.9887654
Lag 4096 0.7677136 0.9851902
Lag 5120 0.7528996 0.9816697

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

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

   edges triangle
  -6.077   -3.616

Individual P-values (lower = worse):
       edges     triangle
1.223728e-09 2.990058e-04
Joint P-value (lower = worse):  0.3997758 .


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



Clearly, somewhere very bad.

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

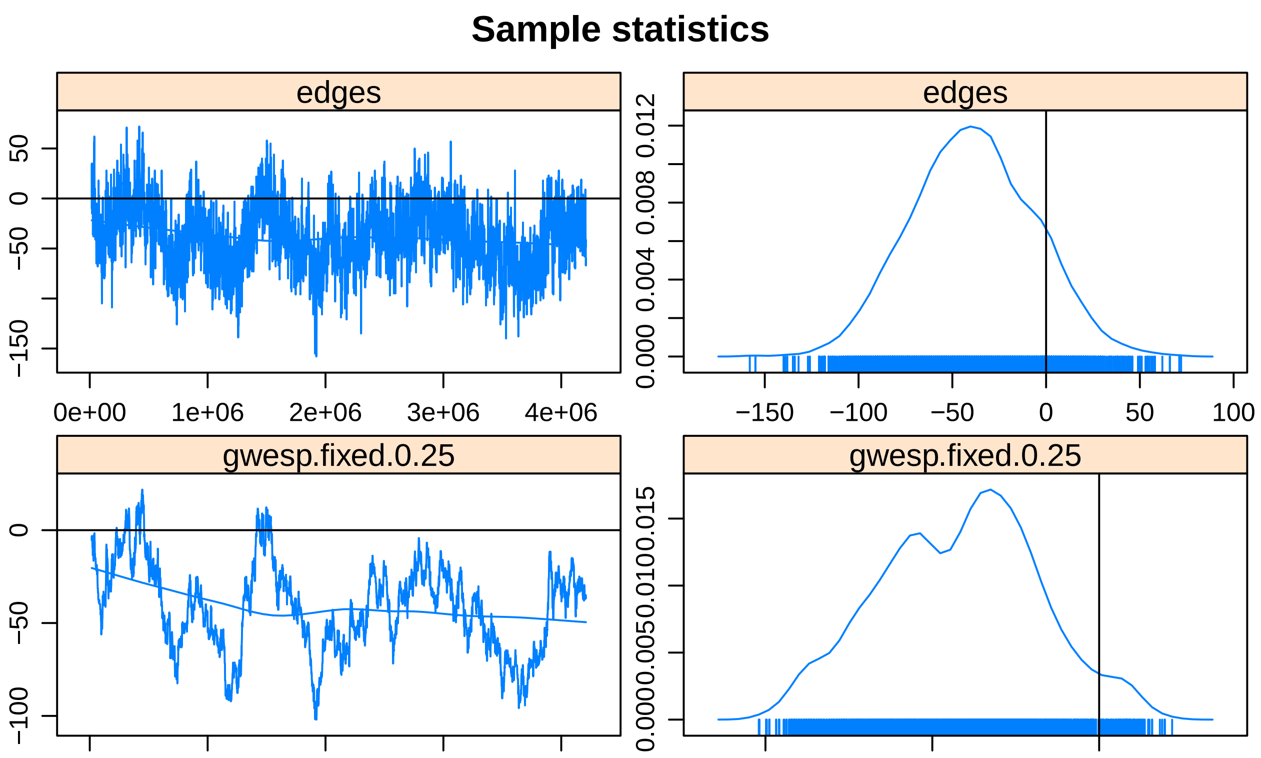
We fix the decay parameter here at 0.25 – it's a reasonable starting place, but in this case we've actually constructed faux.magnolia with this parameter, so we know it's also right.

(NB: You may want to try this and the following ergm calls with verbose output to see the stages of the estimation process:

```
R> fit.mag.02 <- ergm(magnolia ~ edges + gwesp(0.25, fixed = T))

Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 3.304
Iteration 2 of at most 20:
The log-likelihood improved by 0.9182
Step length converged once. Increasing MCMC sample size.
Iteration 3 of at most 20:
The log-likelihood improved by 1.561
Step length converged twice. Stopping.
Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

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

```
R> mcmc.diagnostics(fit.mag.02)

Sample statistics summary:

Iterations = 16384:4209664
```

```
Thinning interval = 1024
Number of chains = 1
Sample size per chain = 4096

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

                   Mean    SD Naive SE Time-series SE
edges            -38.90 32.81   0.5126          3.998
gwesp.fixed.0.25 -41.89 23.71   0.3705          6.455

2. Quantiles for each variable:

                    2.5%    25%    50%   75%  97.5%
edges            -101.00 -61.00 -39.00 -16.0 24.000
gwesp.fixed.0.25  -87.72 -59.16 -39.83 -25.5  6.306


Sample statistics cross-correlations:
                  edges gwesp.fixed.0.25
edges            1.0000           0.6722
gwesp.fixed.0.25 0.6722           1.0000

Sample statistics auto-correlation:
Chain 1
             edges gwesp.fixed.0.25
Lag 0    1.0000000        1.0000000
Lag 1024 0.7555448        0.9934302
Lag 2048 0.6204691        0.9868332
Lag 3072 0.5440855        0.9803922
Lag 4096 0.4973776        0.9742317
Lag 5120 0.4618940        0.9683796

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
           2.665            2.858

Individual P-values (lower = worse):
           edges gwesp.fixed.0.25
     0.007707907      0.004257065
Joint P-value (lower = worse):  0.02953263 .


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



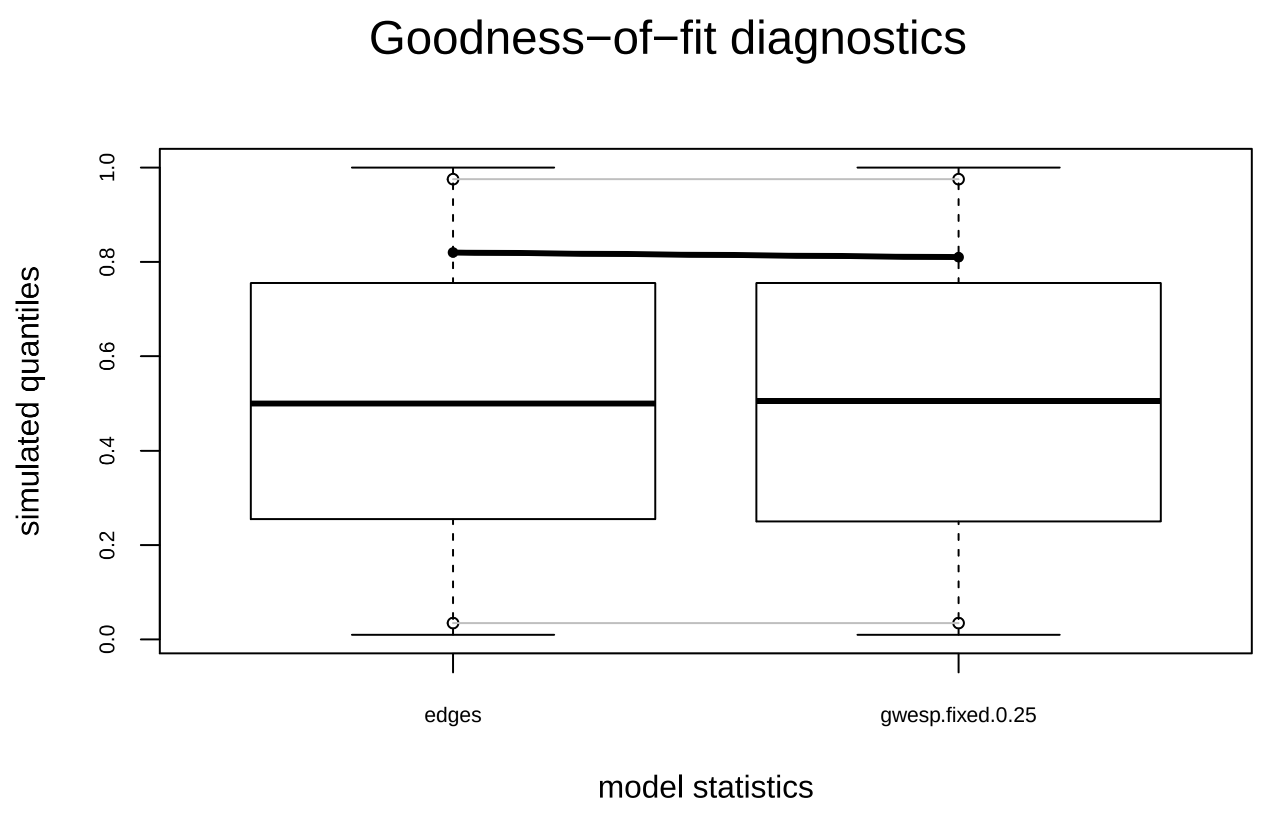
The MCMC diagnostics look somewhat better: the trend is gone, but there's a fair amount of correlation in the samples, and convergence isn't well established. Note the output says:

So let's use gof to check the model statistics for the final estimates. To get both the numerical GOF summaries and the plots, we'll create an object, print it, and then plot it.

```
R> gof.mag.02.model <- gof(fit.mag.02, GOF = ~model)
R> gof.mag.02.model
```

```
Goodness-of-fit for model statistics

                       obs      min      mean       max
edges             974.0000 922.0000 1027.7400 1144.0000
gwesp.fixed.0.25  375.3736 311.7736  427.2849  511.0121
                  MC p-value
edges                   0.36
gwesp.fixed.0.25        0.38
```

```
R> plot(gof.mag.02.model)
```



```
R> fit.mag.03 <- ergm(magnolia ~ edges + gwesp(0.25, fixed = T) +
+     nodematch("Grade") + nodematch("Race") + nodematch("Sex"),
+     control = control.ergm(MCMC.interval = 20000), eval.loglik = F)
```

```
Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 3.514
Iteration 2 of at most 20:
The log-likelihood improved by 0.5666
Step length converged once. Increasing MCMC sample size.
Iteration 3 of at most 20:
The log-likelihood improved by 0.02757
Step length converged twice. Stopping.

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

```
R> summary(fit.mag.03)
```

```

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

Formula:   magnolia ~ edges + gwesp(0.25, fixed = T) + nodematch("Grade") +
    nodematch("Race") + nodematch("Sex")

Iterations:  3 out of 20

Monte Carlo MLE Results:
```

```
                 Estimate Std. Error MCMC % p-value    
edges            -9.78075    0.10678      0  <1e-04 ***
gwesp.fixed.0.25  1.79836    0.04994      0  <1e-04 ***
nodematch.Grade   2.74449    0.08884      0  <1e-04 ***
nodematch.Race    0.90920    0.07422      0  <1e-04 ***
nodematch.Sex     0.76733    0.06267      0  <1e-04 ***
---
Signif. codes:  
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log-likelihood was not estimated for this fit.
To get deviances, AIC, and/or BIC from fit `fit.mag.03` run 
  > fit.mag.03<-logLik(fit.mag.03, add=TRUE)
to add it to the object or rerun this function with eval.loglik=TRUE.
```

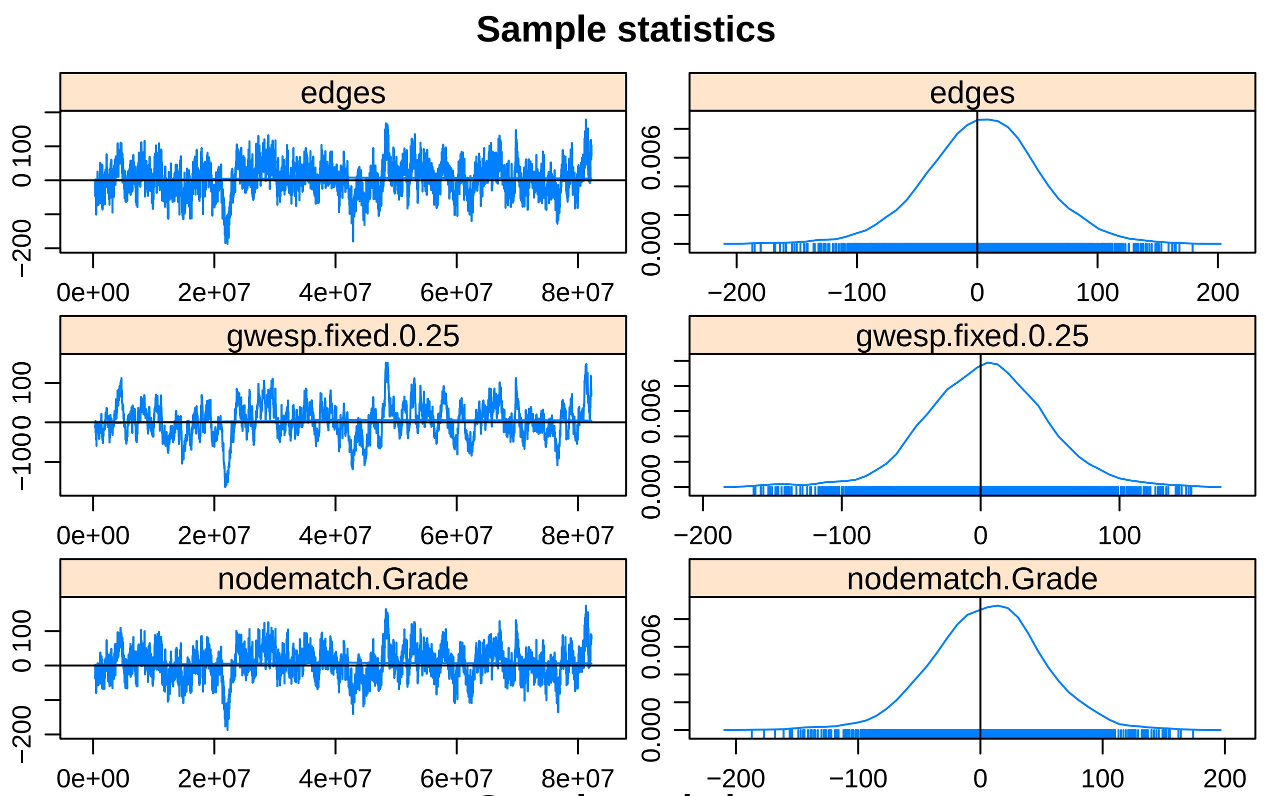
Clearly we're not fitting the model statistics properly, so there's no point in moving on to the GOF for the global statistics. There are two options at this point – one would be to modify the default MCMC parameters to see if this improves convergence, and the other would be to modify the model, since a mis-specified model will also show poor convergence. In this case, we'll do both.

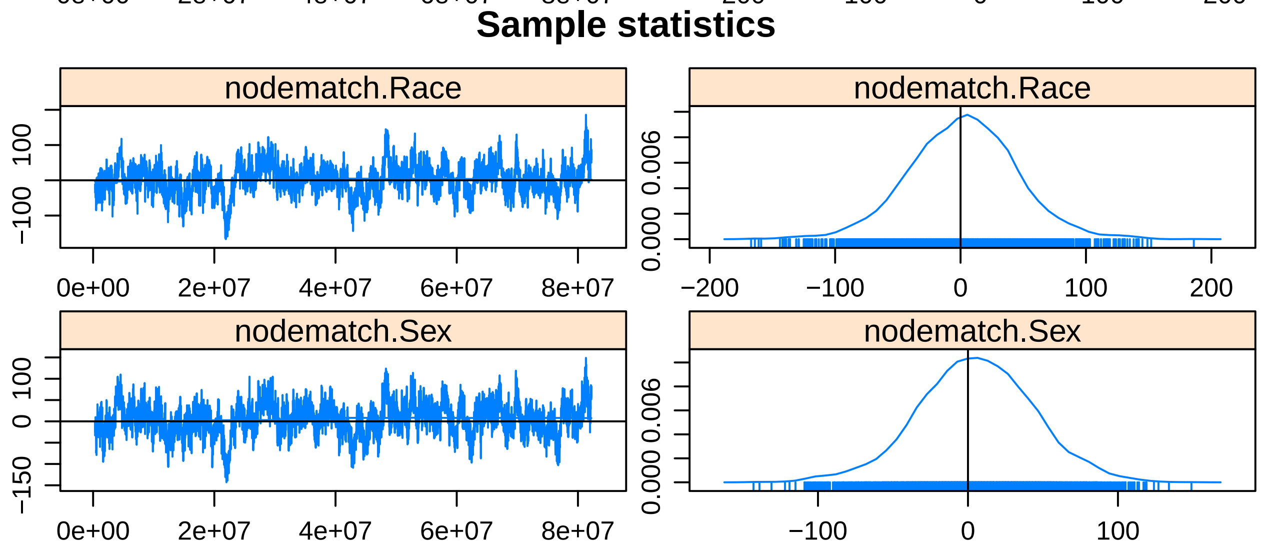
It's reasonable to assume that triad-related processes are not the only factor influencing the pattern of ties we see in the data, so we'll add some terms to the model to capture other dyad-independent effects that would be expected to influence tie status (homophily on the nodal covariates Grade, Race and Sex). In addition, given the serial correlation observed in the MCMC diagnostics from the previous model, we'll also modify the MCMC defaults, and lengthen the interval between the networks selected for the MCMC sample (the default is 1024). The MCMC modifications will increase the estimation time, so we'll cut the log likelihood estimation step.

```
R> mcmc.diagnostics(fit.mag.03)
```

```
Sample statistics summary:

Iterations = 320000:82220000
Thinning interval = 20000 
Number of chains = 1 
Sample size per chain = 4096 

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

                  Mean    SD Naive SE Time-series SE
edges            5.024 48.04   0.7507          4.812
gwesp.fixed.0.25 3.462 43.27   0.6761          5.192
nodematch.Grade  5.538 45.79   0.7154          4.840
nodematch.Race   2.019 43.26   0.6760          4.914
nodematch.Sex    5.216 38.65   0.6040          4.205

2. Quantiles for each variable:

                   2.5%    25%   50%   75% 97.5%
edges            -93.00 -25.00 5.000 36.00 99.62
gwesp.fixed.0.25 -80.38 -24.03 4.279 31.33 88.79
nodematch.Grade  -85.00 -24.00 7.000 35.00 94.00
nodematch.Race   -85.00 -26.00 3.000 30.00 89.00
nodematch.Sex    -74.00 -20.00 5.000 31.00 81.00
```

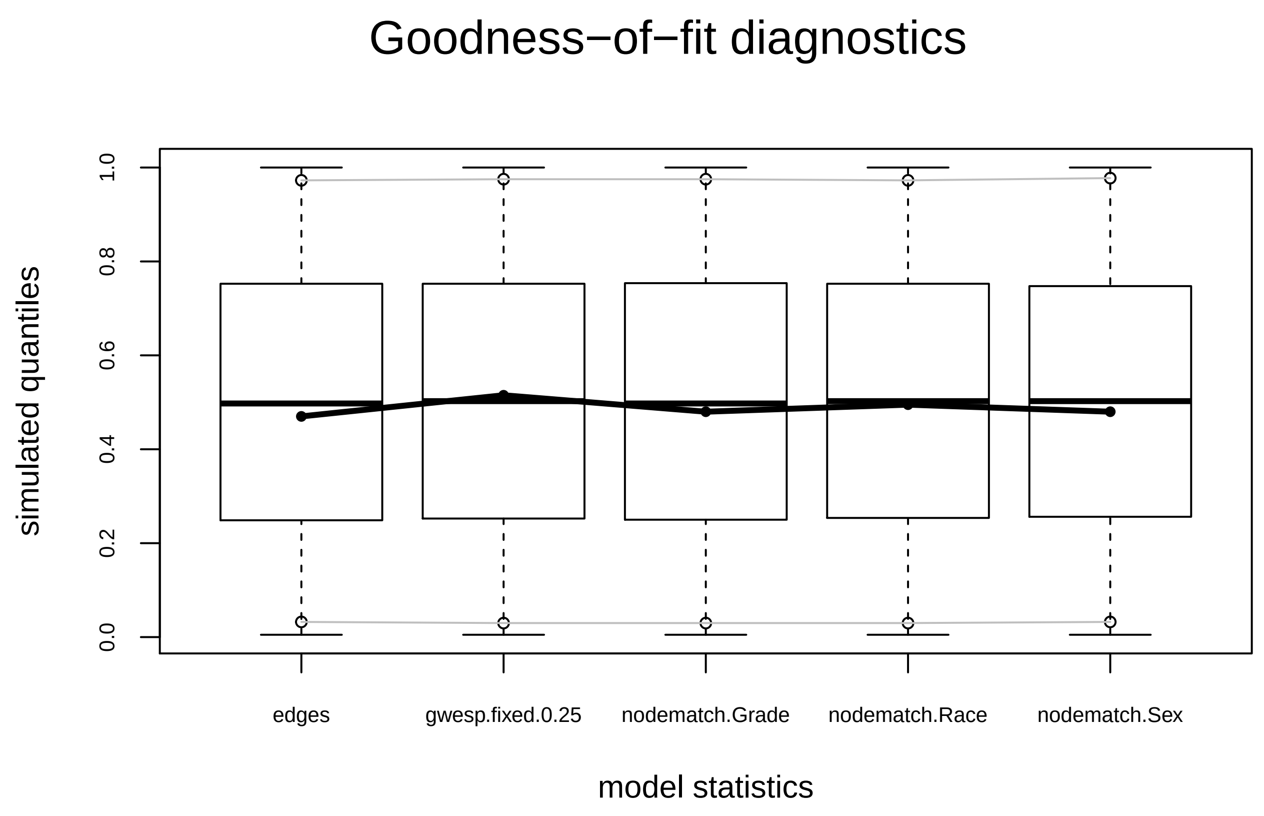
```
Sample statistics cross-correlations:
                      edges gwesp.fixed.0.25 nodematch.Grade
edges             1.0000000        0.8602997       0.9640870
gwesp.fixed.0.25 0.8602997        1.0000000       0.8805791
nodematch.Grade  0.9640870        0.8805791       1.0000000
nodematch.Race   0.9484365        0.8446586       0.9200192
nodematch.Sex    0.9095057        0.8029256       0.8810233
                 nodematch.Race nodematch.Sex
edges                 0.9484365     0.9095057
gwesp.fixed.0.25      0.8446586     0.8029256
nodematch.Grade       0.9200192     0.8810233
nodematch.Race        1.0000000     0.8605918
nodematch.Sex         0.8605918     1.0000000

Sample statistics auto-correlation:
Chain 1
              edges gwesp.fixed.0.25 nodematch.Grade
Lag 0     1.0000000        1.0000000       1.0000000
Lag 20000 0.7295533        0.9592883       0.7777089
Lag 40000 0.6982095        0.9247582       0.7494201
Lag 60000 0.6682060        0.8947391       0.7168942
Lag 80000 0.6457058        0.8660952       0.6970263
Lag 1e+05 0.6312744        0.8384114       0.6797260
          nodematch.Race nodematch.Sex
Lag 0          1.0000000     1.0000000
Lag 20000      0.7480518     0.7186163
Lag 40000      0.7158709     0.6942561
Lag 60000      0.6860887     0.6590477
Lag 80000      0.6587097     0.6385495
Lag 1e+05      0.6446451     0.6291771

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
          0.2709           0.4011           0.3553
  nodematch.Race    nodematch.Sex
          0.1822           0.2233

Individual P-values (lower = worse):
           edges gwesp.fixed.0.25  nodematch.Grade
       0.7864814        0.6883580        0.7223774
  nodematch.Race    nodematch.Sex
       0.8554099        0.8232794
Joint P-value (lower = worse):  0.577677 .


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





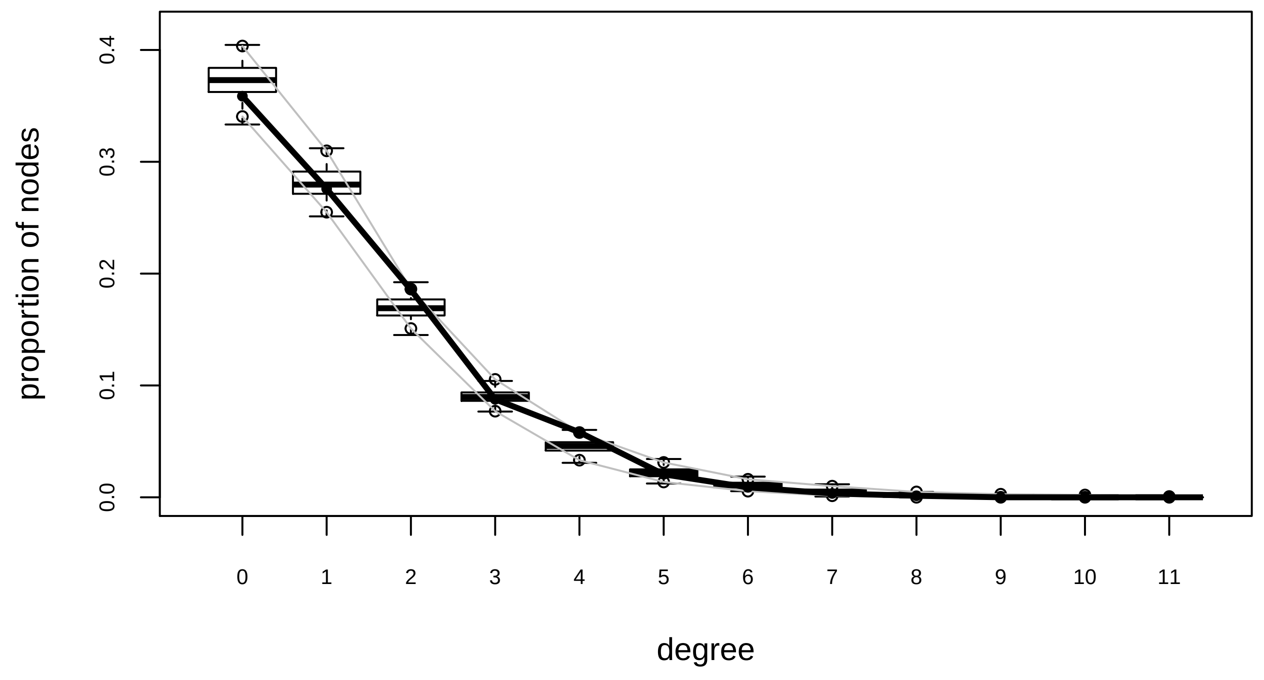
The MCMC diagnostics look much better. Let's take a look at the GOF for model statistics (upping the default number of simulations to 200 from 100).

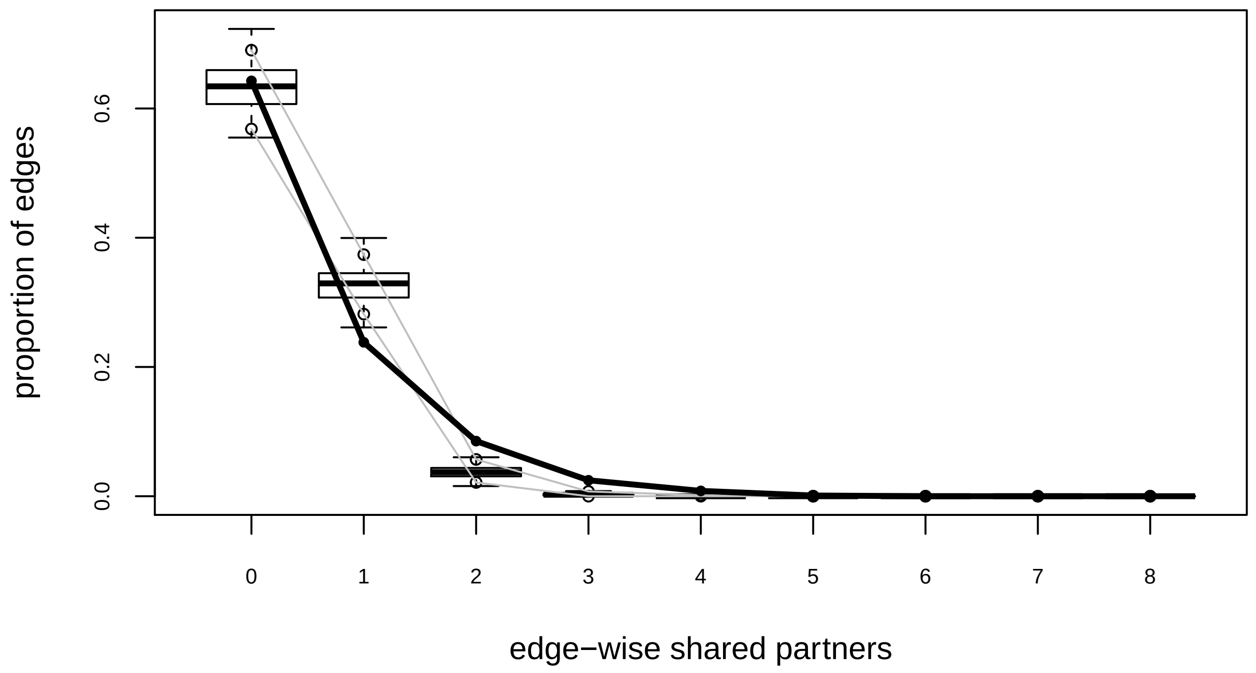
```
R> plot(gof(fit.mag.03, GOF = ~model, control = control.gof.ergm(nsim = 200)))
```

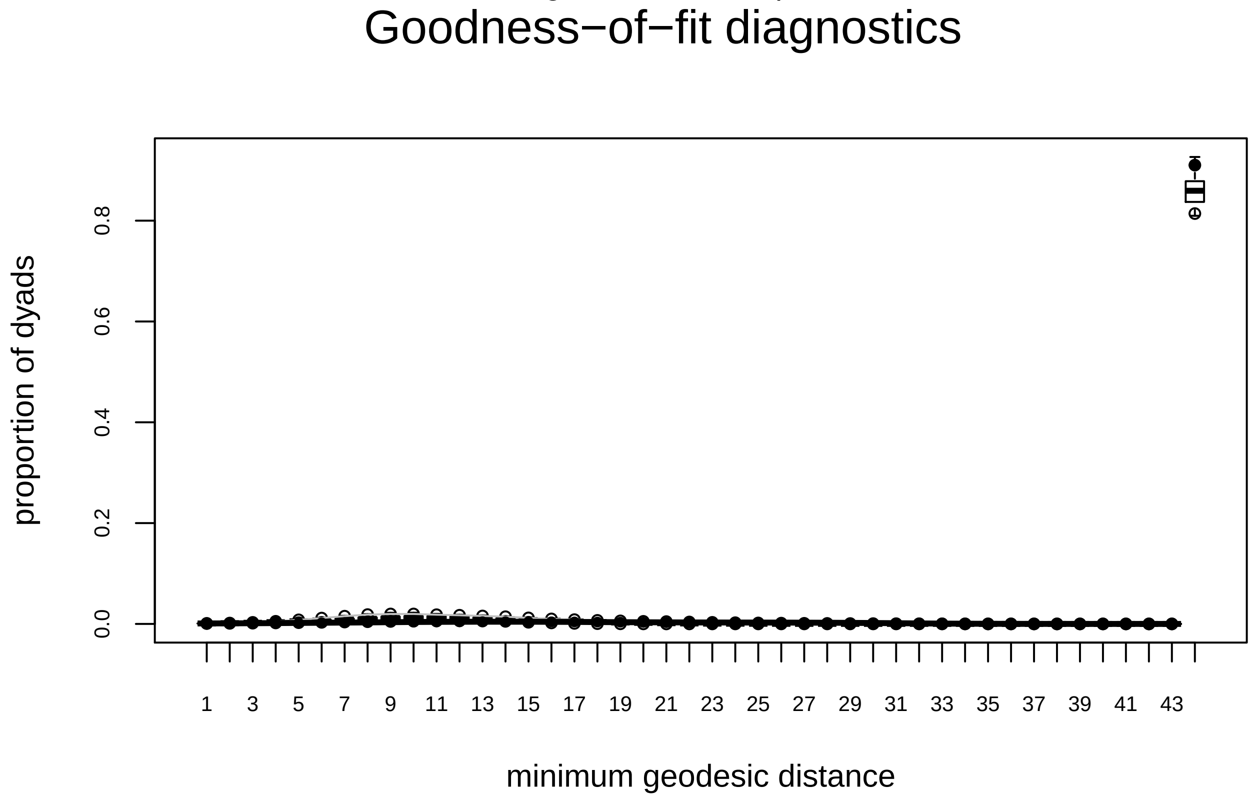


The model statistics look ok, so let's move on to the global GOF statistics.

```
R> plot(gof(fit.mag.03, GOF = ~degree + esp + distance))
```







The global GOF stats look pretty good too. Of course, in real life one might have a lot more trial and error.

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

Statnet Commons: The development group

- Mark S. Handcock handcock@stat.ucla.edu
- David R. Hunter dhunter@stat.psu.edu
- Carter T. Butts buttsc@uci.edu
- Steven M. Goodreau goodreau@u.washington.edu
- Skye Bender-deMoll skyebend@skyeome.net
- Martina Morris morrism@u.washington.edu
- Pavel N. Krivitsky pavel@uow.edu.au

Appendix A: Clarifying the terms – ergm and network

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

ergm

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

network

- *network*: a set of actors and the relations among them. Used interchangeably with the term graph.
- *network package*: one of the packages within the statnet suite; used to create, store, modify and plot the information found in network objects.
- *network object*: a class of object in R used to represent a network.

References

The best primer on the basics of the statnet packages is the special issue of the Journal of Statistical Software v24 (2008): Some of the papers from that issue are noted below.

Goodreau, S., J. Kitts and M. Morris (2009). Birds of a Feather, or Friend of a Friend? Using Statistical Network Analysis to Investigate Adolescent Social Networks. Demography 46(1): 103-125. link

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

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

Handcock, M. S., D. R. Hunter, C. T. Butts, S. M. Goodreau and M. Morris (2008). statnet: Software Tools for the Representation, Visualization, Analysis and Simulation of Network Data. Journal of Statistical Software 42(01).

Hunter DR, Handcock MS, Butts CT, Goodreau SM, Morris M (2008b). ergm: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks. Journal of Statistical Software, 24(3).

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.

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.

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

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