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

Carter T. Butts, University of California, Irvine
David R. Hunter, Penn State University

Joint with the rest of the Statnet Development Team:
Mark S. Handcock, UCLA
Martina Morris, University of Washington
Pavel N. Krivitsky, University of Wollongong
Zack Almquist, University of California, Irvine
Steven M. Goodreau, University of Washington
Skye Bender de-Moll, Oakland

June 22, 2015

Contents

1. Getting Started .......................................................... 3
2. Statistical network modeling; the summary and ergm commands, and supporting functions ... 4
   Bernoulli model ...................................................... 5
   Triad formation ...................................................... 7
   Nodal covariates: effects on mean degree ......................... 8
   Nodal covariates: Homophily ................................... 11
   Directed ties ......................................................... 14
   Missing data ......................................................... 16
3. Model terms available for ergm estimation and simulation ........................................ 20
   Terms provided with ergm ...................................... 20
   Coding new ergm-terms ........................................ 20
4. Network simulation: the simulate command and network.list objects ............................. 21
5. Examining the quality of model fit – GOF ......................................................... 23
6. Diagnostics: troubleshooting and checking for model degeneracy .................................. 29
   What it looks like when a model converges properly ................. 30
   What it looks like when a model fails .................................. 31
7. Working with egocentrically sampled network data .................................................... 43
   Initializing a network ............................................ 44
Target statistics for the model .................................................. 44
ERGM model ........................................................................ 45
Simulation of a complete network from the model ......................... 45
Diagnostics and model re-specification ........................................ 45
8. Additional functionality in the statnet family of packages .......... 53
   Additional functionality ....................................................... 53
   Statnet Commons: The development group ............................ 53
Appendix A: Clarifying the terms – ergm and network .................. 54
References ............................................................................ 54
1. Getting Started

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

To install all of the packages in the statnet suite:

```r
install.packages('statnet')
library(statnet)
```

Or, to only install the specific statnet packages needed for this tutorial:

```r
install.packages('ergm') # will install the network package
install.packages('sna')
```

After the first time, to update the packages one can either repeat the commands above, or use:

```r
update.packages('name.of.package')
```

For this tutorial, we will need one additional package (coda), which is recommended (but not required) by ergm:

```r
install.packages('coda')
```

Make sure the packages are attached:

```r
library(statnet)
```

or

```r
library(ergm)
library(sna)
library(coda)
```

Check package version

```r
# latest versions: ergm 3.1.2 and network 1.9.0 (as of 6/17/2014)
sessionInfo()
```

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

```r
set.seed(0)
```
2. Statistical network modeling; the *summary* and *ergm* commands, and supporting functions

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

The general form for an ERGM can be written as:

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

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

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

\[
\logit (Y_{ij} = 1|y_{ij}^c) = \theta' \delta(y_{ij})
\]

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

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

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

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

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

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

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

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

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

```r
# loads flomarriage and flobusiness data
flomarriage
```

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)
plot(flomarriage, main="Florentine Marriage", cex.main=0.8)  # Plot the flomarriage network
summary(flomarriage~edges)  # Look at the $g(y)$ statistic for this model

edges

20

flomodel.01 <- ergm(flomarriage~edges)  # Estimate the model
summary(flomodel.01)  # The fitted model object

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

Formula:  flomarriage ~ edges

Iterations:  5 out of 20

Monte Carlo MLE Results:

<table>
<thead>
<tr>
<th></th>
<th>Estimate</th>
<th>Std. Error</th>
<th>MCMC %</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>-1.6094</td>
<td>0.2449</td>
<td>0</td>
<td>&lt;1e-04  ***</td>
</tr>
</tbody>
</table>

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.)

Figure 1: plot of chunk unnamed-chunk-10

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

$$
= -1.609 \times \text{change in the number of ties}
= -1.609 \times 1
$$
for all ties, since the addition of any tie to the network always changes the number of ties by 1 for a tie toggled from 0 to 1 (or by -1 for a tie toggled from 1 to 0).

The corresponding probability is:

$$
= \exp(-1.609)/(1 + \exp(-1.609))
= 0.1667
$$

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

Triad formation

Let’s add a term often thought to be a measure of “clustering”: the number of completed triangles. The \textit{ergm-term} for this is \textbf{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 0.004497
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 0.004005
Step length converged twice. Stopping.

This model was fit using MCMC. To examine model diagnostics and check for degeneracy, use the \textsf{mcmc.diagnostics()} function.

```r
summary(flomodel.02)
```
Now, how should we interpret coefficients?
The conditional log-odds of two actors having a tie is:

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

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

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

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

```
edges triangle
-1.6608684  0.1415158
```

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

```
     Min. 1st Qu.  Median    Mean 3rd Qu. Max. 
3.000  17.500  39.000   42.560  48.250  146.00
```
Figure 2: plot of chunk unnamed-chunk-14
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'))
summary(flomodel.03)

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

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

Iterations:  4 out of 20

Monte Carlo MLE Results:

         Estimate Std. Error MCMC % p-value
edges    -2.594929  0.536056        0 <1e-04  ***
nodecov.wealth  0.010546  0.004674        0  0.0259    *

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

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

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

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

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

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

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

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

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

10
Nodal covariates: Homophily

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

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

mesa

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

Vertex attribute names:
Grade Race Sex

No edge attributes

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

fauxmodel.01 <- ergm(mesa ~edges + nodematch('Grade',diff=T) + nodematch('Race',diff=T))

Observed statistic(s) nodematch.Race.Black and nodematch.Race.Other are at their smallest attainable values. Their

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:

<table>
<thead>
<tr>
<th>Estimate</th>
<th>Std. Error</th>
<th>MCMC %</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>-6.2328</td>
<td>0.1742</td>
<td>0 &lt;1e-04 ***</td>
</tr>
<tr>
<td>nodematch.Grade.7</td>
<td>2.8740</td>
<td>0.1981</td>
<td>0 &lt;1e-04 ***</td>
</tr>
<tr>
<td>nodematch.Grade.8</td>
<td>2.8788</td>
<td>0.2391</td>
<td>0 &lt;1e-04 ***</td>
</tr>
<tr>
<td>nodematch.Grade.9</td>
<td>2.4642</td>
<td>0.2647</td>
<td>0 &lt;1e-04 ***</td>
</tr>
<tr>
<td>nodematch.Grade.10</td>
<td>2.5692</td>
<td>0.3770</td>
<td>0 &lt;1e-04 ***</td>
</tr>
<tr>
<td>nodematch.Grade.11</td>
<td>3.2921</td>
<td>0.2978</td>
<td>0 &lt;1e-04 ***</td>
</tr>
<tr>
<td>nodematch.Grade.12</td>
<td>3.8376</td>
<td>0.4592</td>
<td>0 &lt;1e-04 ***</td>
</tr>
<tr>
<td>nodematch.Race.Black</td>
<td>-Inf</td>
<td>0.0000</td>
<td>0 &lt;1e-04 ***</td>
</tr>
</tbody>
</table>
Figure 3: plot of chunk unnamed-chunk-16
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:

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

```
table(mesa %>% 'Race') # Frequencies of race
```

<table>
<thead>
<tr>
<th>Black</th>
<th>Hisp</th>
<th>NatAm</th>
<th>Other</th>
<th>White</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>109</td>
<td>68</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

```
Black   Hisp   NatAm  Other  White
0       8      13     0      5
Hisp    8      53     41     1    22
NatAm   13     41     46     0    10
Other   0      1      0      0    0
White   5      22     10     0    4
```

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

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

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

```
edges   nodematch.Grade.7   nodematch.Grade.8
203      75                  33
nodematch.Grade.9   nodematch.Grade.10   nodematch.Grade.11
23       9                   17
6        0                   53
46       0                   4
```

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

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

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

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

samplk3

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

Vertex attribute names:
cloisterville group vertex.names
No edge attributes

plot(samplk3)

summary(samplk3~edges+mutual)

edges mutual
56 15

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

sampmodel.01 <- ergm(samplk3~edges+mutual)
```
Figure 4: plot of chunk unnamed-chunk-20
Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 0.001635
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 0.002764
Step length converged twice. Stopping.

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

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

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

Formula: samplk3 ~ edges + mutual

Iterations: 2 out of 20

Monte Carlo MLE Results:

<table>
<thead>
<tr>
<th></th>
<th>Estimate</th>
<th>Std. Error</th>
<th>MCMC %</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>-2.1572</td>
<td>0.2186</td>
<td>0</td>
<td>&lt;1e-04 ***</td>
</tr>
<tr>
<td>mutual</td>
<td>2.3041</td>
<td>0.4865</td>
<td>0</td>
<td>&lt;1e-04 ***</td>
</tr>
</tbody>
</table>

---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

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

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

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

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

Missing data

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

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

```
missnet <- network.initialize(10,directed=F)
summary(misssnet)
```

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 1 0 0 0 0
3 0 0 0 0 0 1 0 0 0 0
4 0 0 0 0 NA 0 0 NA 0 0
5 0 0 0 0 NA 0 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 edge colored red.
tempnet <- missnet
missnetmat <- as.matrix(missnet)
missnetmat[is.na(missnetmat)] <- 2
plot(tempnet,label = network.vertex.names(tempnet),edge.col = missnetmat)

summary(missnet~edges)

edges
3

summary(ergm(missnet~edges))

===================================
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
Figure 5: plot of chunk unnamed-chunk-22
The coefficient equals -2.56, which corresponds to a probability of 7.14%. Our network has 3 ties, out of the 42 non-missing nodal pairs (10 choose 2 minus 3): 3/42 = 7.14%. So our estimate represents the probability of a tie in the observed sample.

Now let’s assign those missing ties the value “0” and see what happens.

```r
missnet_bad <- missnet
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 1 0 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(ergm(missnet_bad ~ edges))
```

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

Formula: missnet_bad ~ edges

Iterations: 5 out of 20

Monte Carlo MLE Results:
  Estimate Std. Error MCMC % p-value
```
null deviance: 62.38 on 45 degrees of freedom
residual deviance: 22.04 on 44 degrees of freedom

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

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

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

3. Model terms available for **ergm** estimation and simulation

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

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

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

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

**Terms provided with ergm**

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

```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**

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

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

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

```r
flomodel.03.sim <- simulate(flomodel.03, nsim=10)
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,] 21  2547
  [2,] 17  2150
  [3,] 21  2468
  [4,] 30  3499
  [5,] 20  2441
  [6,] 19  2112
  [7,] 20  2396
  [8,] 16  1555
  [9,] 14  1585
 [10,] 15  1800
```

```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= 21
  missing edges= 0
  non-missing edges= 21
```

Vertex attribute names:
```
  priorates totalties vertex.names wealth
```

No edge attributes
Figure 6: plot of chunk unnamed-chunk-25
Voila. Of course, yours will look somewhat different.

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

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

## 5. Examining the quality of model fit – GOF

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

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

The `gof` function is a bit different than the `summary`, `ergm`, and `simulate` functions, in that it currently only takes 3 ergm-terms as arguments: degree, esp (edgewise 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).

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

### Goodness-of-fit for degree

<table>
<thead>
<tr>
<th>obs</th>
<th>min</th>
<th>min</th>
<th>max</th>
<th>MC</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1.47</td>
<td>5</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>0</td>
<td>3.49</td>
<td>9</td>
<td>0.88</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4.31</td>
<td>8</td>
<td>0.34</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>0</td>
<td>3.20</td>
<td>8</td>
<td>0.14</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1.88</td>
<td>6</td>
<td>1.00</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0.95</td>
<td>4</td>
<td>0.68</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0.40</td>
<td>3</td>
<td>0.68</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0.18</td>
<td>2</td>
<td>1.00</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0.07</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0.04</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>1</td>
<td>1.00</td>
</tr>
</tbody>
</table>

### Goodness-of-fit for edgewise shared partner

<table>
<thead>
<tr>
<th>esp0</th>
<th>obs</th>
<th>min</th>
<th>min</th>
<th>max</th>
<th>MC</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
<td>3</td>
<td>12.01</td>
<td>23</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>esp1</td>
<td>7</td>
<td>0</td>
<td>5.71</td>
<td>16</td>
<td>0.82</td>
<td></td>
</tr>
<tr>
<td>esp2</td>
<td>1</td>
<td>0</td>
<td>1.31</td>
<td>7</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>esp3</td>
<td>0</td>
<td>0</td>
<td>0.24</td>
<td>8</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>esp4</td>
<td>0</td>
<td>0</td>
<td>0.05</td>
<td>2</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>esp5</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>1</td>
<td>1.00</td>
<td></td>
</tr>
</tbody>
</table>
Goodness-of-fit for minimum geodesic distance

<table>
<thead>
<tr>
<th>obs</th>
<th>min</th>
<th>mean</th>
<th>max</th>
<th>MC</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>11</td>
<td>19.33</td>
<td>28</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>16</td>
<td>32.95</td>
<td>60</td>
<td>0.90</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>5</td>
<td>25.82</td>
<td>41</td>
<td>0.46</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>0</td>
<td>10.74</td>
<td>23</td>
<td>0.52</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>0</td>
<td>3.04</td>
<td>15</td>
<td>0.82</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0.70</td>
<td>10</td>
<td>1.00</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0.22</td>
<td>9</td>
<td>1.00</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0.09</td>
<td>6</td>
<td>1.00</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>2</td>
<td>1.00</td>
</tr>
<tr>
<td>Inf</td>
<td>15</td>
<td>0</td>
<td>27.09</td>
<td>82</td>
<td>0.94</td>
</tr>
</tbody>
</table>

plot(flomodel.03.gof)
mesamodel.02 <- ergm(mesa-edges)
mesamodel.02.gof <- gof(mesamodel.02~degree + esp + distance, nsim=10)
plot(mesamodel.02.gof)
6. Diagnostics: troubleshooting and checking for model degeneracy

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

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

For more detailed discussion of model degeneracy in the ERGM context, see the papers by Mark Handcock referenced below.
In the worst case scenario, we end up not being able to obtain coefficient 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.

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

edges degree1
    15     3

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

Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 0.2358
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 0.001153
Step length converged twice. Stopping.

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

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

Sample statistics summary:

- Iterations = 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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>SD</th>
<th>Naive SE</th>
<th>Time-series SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>0.16895</td>
<td>3.741</td>
<td>0.05845</td>
<td>0.06060</td>
</tr>
<tr>
<td>degree1</td>
<td>-0.05688</td>
<td>1.631</td>
<td>0.02549</td>
<td>0.02609</td>
</tr>
</tbody>
</table>

2. Quantiles for each variable:

<table>
<thead>
<tr>
<th>Variable</th>
<th>2.5%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>-7</td>
<td>-2</td>
<td>0</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>degree1</td>
<td>-3</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Sample statistics cross-correlations:

<table>
<thead>
<tr>
<th>edges</th>
<th>degree1</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>1.000000</td>
</tr>
<tr>
<td>degree1</td>
<td>-0.449534</td>
</tr>
</tbody>
</table>
Sample statistics auto-correlation:
Chain 1

<table>
<thead>
<tr>
<th>Lag</th>
<th>edges</th>
<th>degree1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000000000</td>
<td>1.00000000</td>
</tr>
<tr>
<td>1024</td>
<td>-0.002063692</td>
<td>-0.01023343</td>
</tr>
<tr>
<td>2048</td>
<td>0.037775537</td>
<td>0.03344223</td>
</tr>
<tr>
<td>3072</td>
<td>-0.007354148</td>
<td>0.01417543</td>
</tr>
<tr>
<td>4096</td>
<td>-0.003838989</td>
<td>0.01389468</td>
</tr>
<tr>
<td>5120</td>
<td>-0.004162156</td>
<td>0.01452616</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>edges</th>
<th>degree1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7137</td>
<td>0.4301</td>
</tr>
</tbody>
</table>

Individual P-values (lower = worse):
<table>
<thead>
<tr>
<th>edges</th>
<th>degree1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4754184</td>
<td>0.6671138</td>
</tr>
</tbody>
</table>

Joint P-value (lower = worse): 0.5390232

Loading required namespace: latticeExtra

Recent changes in the ergm estimation algorithm mean that these plots can no longer be used to ensure that the mean

...the observed network statistics. For that functionality, please use the GOF command: gof(ergmFitObject, GOF=~model).

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

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

What it looks like when a model fails

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

```r
data(’faux.magnolia.high’) magnolia <- faux.magnolia.high plot(magnolia, vertex.cex=.5) summary(magnolia~edges+triangle)
```
Figure 7: plot of chunk unnamed-chunk-28
Figure 8: plot of chunk unnamed-chunk-30
edges triangle
  974   169

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

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

Error: Number of edges in a simulated network exceeds that in the observed by a factor of more than 20. This is a strong indicator of a degenerate model specification that the algorithm stops after 3 iterations, to avoid heading off into areas that would cause memory issues. If you'd like to peek a bit more under the hood, you can stop the algorithm earlier to catch where it's heading:

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

Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 2:
The log-likelihood improved by 3.135
Iteration 2 of at most 2:
The log-likelihood improved by 1.59
Step length converged once. Increasing MCMC sample size.

MCMLE estimation did not converge after 2 iterations. The estimated coefficients may not be accurate. Estimation may be resumed by passing the coefficients as initial values; see /quotes\single`init`/quotes\single under ?control.ergm for details.

This model was fit using MCMC. To examine model diagnostics and check for degeneracy, use the mcmc.diagnostics() function:

mcmc.diagnostics(fit)

Clearly, somewhere very bad.

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

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

Evaluating network in model
Initializing Metropolis-Hastings proposal(s): ergm:MH_TNT
Initializing model.
Using initial method 'MPLE'.
Fitting initial model.
MPLE covariate matrix has 85 rows.
Fitting ERGM.
Starting maximum likelihood estimation via MCMLE:
Density guard set to 19563 from an initial count of 974 edges.
Iteration 1 of at most 20 with parameter:
  edges gwesp.fixed.0.25
    -7.350244   2.147119
Sampler accepted 69.660% of 1048576 proposed steps.
Figure 9: plot of chunk unnamed-chunk-33
Sample size = 1024 by 1024
Back from unconstrained MCMC. Average statistics:
   edges gwesp.fixed.0.25
   9.197266 -73.801705
Average estimating equation values:
   edges gwesp.fixed.0.25
   9.197266 -73.801705
Calling MCMLE Optimization...
Using Newton-Raphson Step with step length 0.461046695709229 ...
Using lognormal metric (see control.ergm function).
Optimizing loglikelihood
The log-likelihood improved by 2.668
Iteration 2 of at most 20 with parameter:
   edges gwesp.fixed.0.25
   -7.394993 2.298363
Sampler accepted 62.532% of 1048576 proposed steps.
Sample size = 1024 by 1024
Back from unconstrained MCMC. Average statistics:
   edges gwesp.fixed.0.25
   93.55957 25.39038
Average estimating equation values:
   edges gwesp.fixed.0.25
   93.55957 25.39038
Calling MCMLE Optimization...
Using Newton-Raphson Step with step length 1 ...
Using lognormal metric (see control.ergm function).
Optimizing loglikelihood
Starting MCMC s.e. computation.
The log-likelihood improved by 1.849
Step length converged once. Increasing MCMC sample size.
Iteration 3 of at most 20 with parameter:
   edges gwesp.fixed.0.25
   -7.46717 2.36185
Sampler accepted 56.404% of 4194304 proposed steps.
Sample size = 4096 by 4096
Back from unconstrained MCMC. Average statistics:
   edges gwesp.fixed.0.25
   109.7241 105.4302
Average estimating equation values:
   edges gwesp.fixed.0.25
   109.7241 105.4302
Calling MCMLE Optimization...
Using Newton-Raphson Step with step length 0.75786440819502 ...
Using lognormal metric (see control.ergm function).
Optimizing loglikelihood
Starting MCMC s.e. computation.
The log-likelihood improved by 2.051
Iteration 4 of at most 20 with parameter:
   edges gwesp.fixed.0.25
   -7.480477 2.324368
Sampler accepted 60.833% of 1048576 proposed steps.
Sample size = 1024 by 1024
Back from unconstrained MCMC. Average statistics:
   edges gwesp.fixed.0.25
   18.85059 22.88774
Average estimating equation values:
   edges gwesp.fixed.0.25
   18.85059 22.88774
Calling MCMLE Optimization...
Using Newton-Raphson Step with step length 1 ... 
Using lognormal metric (see control.ergm function).
Optimizing loglikelihood
Starting MCMC s.e. computation.
The log-likelihood improved by 0.06051
Step length converged once. Increasing MCMC sample size.
Iteration 5 of at most 20 with parameter:
    edges gwesp.fixed.0.25
-7.475185  2.314722
Sampler accepted  62.712% of 4194304 proposed steps.
Sample size = 4096 by 4096
Back from unconstrained MCMC. Average statistics:
    edges gwesp.fixed.0.25
-8.526635  -9.865733
Average estimating equation values:
    edges gwesp.fixed.0.25
-8.526635  -9.865733
Calling MCMLE Optimization...
Using Newton-Raphson Step with step length 1 ... 
Using lognormal metric (see control.ergm function).
Optimizing loglikelihood
Starting MCMC s.e. computation.
The log-likelihood improved by 0.05209
Step length converged twice. Stopping.
Evaluating log-likelihood at the estimate.

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

```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:

   Variate Mean    SD    Naive SE  Time-series SE
   edges   -8.526  39.34 0.6148 9.197
   gwesp.fixed.0.25  -9.866 30.99 0.4842 11.140

2. Quantiles for each variable:

   Variate 2.5% 25%  50%  75% 97.5%
   edges  -86.62 -35.0 -9.00 19.00 67.62
   gwesp.fixed.0.25 -69.60 -31.5 -12.06 13.99 50.17

Sample statistics cross-correlations:

   edges gwesp.fixed.0.25
   edges  1.0000000 0.7839473
   gwesp.fixed.0.25 0.7839473 1.0000000

Sample statistics auto-correlation:
Chain 1
   edges gwesp.fixed.0.25
Sample statistics burn-in diagnostic (Geweke):
Chain 1
Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

Individual P-values (lower = worse):
edges gwesp.fixed.0.25
0.7229846 0.7026075
Joint P-value (lower = worse): 0.6762997.

Recent changes in the ergm estimation algorithm mean that these plots can no longer be used to ensure that the mean

Better, but not great. Here we’ll change two things – add some more reasonable terms to the model, and add in some

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

Evaluating network in model
Initializing Metropolis-Hastings proposal(s): ergm:MH_TNT
Initializing model.
Using initial method 'MPLE'.
Fitting initial model.
MPLE covariate matrix has 211 rows.
Fitting ERGM.
Starting maximum likelihood estimation via MCMLE:
Density guard set to 19563 from an initial count of 974 edges.
Iteration 1 of at most 20 with parameter:
edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
-9.8619687 1.6946112 2.8534613 0.9886313
nodematch.Sex
0.8245285
Sampler accepted 30.343% of 50000000 proposed steps.
Sample size = 50000 by 50000
Back from unconstrained MCMC. Average statistics:
edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
110.90656 50.47711 109.28718 106.84220
nodematch.Sex
97.29466
Average estimating equation values:
edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
110.90656 50.47711 109.28718 106.84220
nodematch.Sex
Figure 10: plot of chunk unnamed-chunk-34
Calling MCMLE Optimization...
Using Newton-Raphson Step with step length 1 ...
Using lognormal metric (see control.ergm function).
Optimizing loglikelihood
Starting MCMC s.e. computation.
The log-likelihood improved by 5.142
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20 with parameter:
edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
   -9.7795529 1.8047511 2.7533827 0.9100361
   0.7664625
Sampler accepted 31.413% of 200000000 proposed steps.
Sample size = 200000 by 2e+05
Back from unconstrained MCMC. Average statistics:
edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
   28.31712 25.26616 27.52406 25.82301
   22.48074
Average estimating equation values:
edges gwesp.fixed.0.25 nodematch.Grade nodematch.Race
   28.31712 25.26616 27.52406 25.82301
   22.48074
Calling MCMLE Optimization...
Using Newton-Raphson Step with step length 1 ...
Using lognormal metric (see control.ergm function).
Optimizing loglikelihood
Starting MCMC s.e. computation.
The log-likelihood improved by 0.198
Step length converged twice. Stopping.
Evaluating log-likelihood at the estimate.

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

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

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>SD</th>
<th>Naive SE</th>
<th>Time-series SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>28.32</td>
<td>47.21</td>
<td>0.10556</td>
<td>2.295</td>
</tr>
<tr>
<td>gwesp.fixed.0.25</td>
<td>25.27</td>
<td>42.70</td>
<td>0.09548</td>
<td>3.252</td>
</tr>
<tr>
<td>nodematch.Grade</td>
<td>27.52</td>
<td>45.17</td>
<td>0.10100</td>
<td>2.421</td>
</tr>
<tr>
<td>nodematch.Race</td>
<td>25.82</td>
<td>43.54</td>
<td>0.09736</td>
<td>2.260</td>
</tr>
<tr>
<td>nodematch.Sex</td>
<td>22.48</td>
<td>38.49</td>
<td>0.08607</td>
<td>1.855</td>
</tr>
</tbody>
</table>

2. Quantiles for each variable:

<table>
<thead>
<tr>
<th>Variable</th>
<th>2.5%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>-65.00</td>
<td>-3.00</td>
<td>29.00</td>
<td>60.00</td>
<td>120.0</td>
</tr>
</tbody>
</table>
Sample statistics cross-correlations:

<table>
<thead>
<tr>
<th></th>
<th>edges</th>
<th>gwesp.fixed.0.25</th>
<th>nodematch.Grade</th>
<th>nodematch.Race</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>1.0000000</td>
<td>0.8523791</td>
<td>0.9612456</td>
<td>0.9438962</td>
</tr>
<tr>
<td>gwesp.fixed.0.25</td>
<td>0.8523791</td>
<td>1.0000000</td>
<td>0.8721761</td>
<td>0.8455619</td>
</tr>
<tr>
<td>nodematch.Grade</td>
<td>0.9612456</td>
<td>0.8721761</td>
<td>1.0000000</td>
<td>0.9156183</td>
</tr>
<tr>
<td>nodematch.Race</td>
<td>0.9438962</td>
<td>0.8455619</td>
<td>0.9156183</td>
<td>1.0000000</td>
</tr>
<tr>
<td>nodematch.Sex</td>
<td>0.9076444</td>
<td>0.8003289</td>
<td>0.8763622</td>
<td>0.8589487</td>
</tr>
</tbody>
</table>

Sample statistics auto-correlation:

Chain 1

<table>
<thead>
<tr>
<th></th>
<th>edges</th>
<th>gwesp.fixed.0.25</th>
<th>nodematch.Grade</th>
<th>nodematch.Race</th>
<th>nodematch.Sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lag 0</td>
<td>1.0000000</td>
<td>1.0000000</td>
<td>1.0000000</td>
<td>1.0000000</td>
<td>1.0000000</td>
</tr>
<tr>
<td>Lag 1000</td>
<td>0.9404781</td>
<td>0.9976127</td>
<td>0.9627409</td>
<td>0.9537489</td>
<td></td>
</tr>
<tr>
<td>Lag 2000</td>
<td>0.8973518</td>
<td>0.9952788</td>
<td>0.9330657</td>
<td>0.9194953</td>
<td></td>
</tr>
<tr>
<td>Lag 3000</td>
<td>0.8658492</td>
<td>0.9924930</td>
<td>0.9089265</td>
<td>0.8935034</td>
<td></td>
</tr>
<tr>
<td>Lag 4000</td>
<td>0.8420106</td>
<td>0.9907723</td>
<td>0.8890070</td>
<td>0.8730902</td>
<td></td>
</tr>
<tr>
<td>Lag 5000</td>
<td>0.8230110</td>
<td>0.9885899</td>
<td>0.8723313</td>
<td>0.8563747</td>
<td></td>
</tr>
</tbody>
</table>

Sample statistics burn-in diagnostic (Geweke):

Chain 1

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

<table>
<thead>
<tr>
<th></th>
<th>edges</th>
<th>gwesp.fixed.0.25</th>
<th>nodematch.Grade</th>
<th>nodematch.Race</th>
<th>nodematch.Sex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.761</td>
<td>2.133</td>
<td>2.667</td>
<td>3.059</td>
<td>2.777</td>
</tr>
</tbody>
</table>

Individual P-values (lower = worse):

<table>
<thead>
<tr>
<th></th>
<th>edges</th>
<th>gwesp.fixed.0.25</th>
<th>nodematch.Grade</th>
<th>nodematch.Race</th>
<th>nodematch.Sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges gwesp.fixed.0.25</td>
<td>0.005761721</td>
<td>0.032929406</td>
<td>0.007654797</td>
<td>0.002221343</td>
<td></td>
</tr>
<tr>
<td>nodematch.Grade</td>
<td>0.005486245</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodematch.Race</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodematch.Sex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Recent changes in the ergm estimation algorithm mean that these plots can no longer be used to ensure that the mean

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

**MORAL:** Degeneracy is an indicator of a poorly specified model. It is not a property of all ERGMs, but it is

associated with some dyadic-dependent terms, in particular, the reduced homogenous Markov specifications (e.g.,

2-stars and triangle terms). For a good technical discussion of unstable terms see Schweinberger 2012. For a discussion

of alternative terms that exhibit more stable behavior see Snijders et al. 2006. and for the gwesp term (and the
curved exponential family terms in general) see Hunter and Handcock 2006.

7. Working with egocentrically sampled network data

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

sampled data.

In many empirical contexts, it is not feasible to collect a network census or even an adaptive (link-traced) sample. Even

when one of these may be possible in practice, egocentrically sampled data are typically cheaper and easier to
collect.

Long regarded as the poor country cousin in the network data family, egocentric data contain a remarkable amount

of information. With the right statistical methods, such data can be used to explore the properties of the complete

networks in which they are embedded. The basic idea here is to combine what is observed, with assumptions, to
define a class of models that describe the distribution of networks that are centered on the observed properties. The
variation in these networks quantifies some of the uncertainty introduced by the assumptions.

Let’s start with a simple fictional example: You have a sample of persons who were asked about the friends they had
seen face-to-face more than once in the last week. The respondent was asked their own sex, and the sex of each friend
(for up to three friends). Summary statistics from these data thus include the sex distribution, the degree distribution
(it could be broken down by sex, but we will just examine the marginal distribution here), and the joint distribution
of the respondent and friend’s sex (the sex mixing matrix). Let’s assume there are equal numbers of men and women
in the sampled respondents. The other distributions are shown below:

<table>
<thead>
<tr>
<th>Degree</th>
<th>Frequency</th>
<th>Fraction</th>
<th>Ties</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>180</td>
<td>0.36</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>245</td>
<td>0.49</td>
<td>245</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>0.12</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>0.03</td>
<td>45</td>
</tr>
<tr>
<td>Total</td>
<td>500</td>
<td>1.00</td>
<td>410</td>
</tr>
</tbody>
</table>

Sex mixing matrix (410 total)

<table>
<thead>
<tr>
<th>Friend</th>
<th>M</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Respondent</td>
<td>164</td>
<td>44</td>
</tr>
<tr>
<td>F</td>
<td>26</td>
<td>176</td>
</tr>
</tbody>
</table>

So, total N respondents = 500, total N friends reported = 410.

We can use an ERGM to fit the parameters associated with these observed statistics, then use the fitted model to
simulate complete networks that are drawn from the distribution of networks that is centered around these statistics.
As a theoretical exercise, this provides a method for investigating the complete network implications of these observed
summary statistics. As an empirical exercise, the primary assumption needed for inference is that the data we have is
sampled from a population in equilibrium (and, as in all statistical inference, that our model is correct). The theory
for this is developed in Krivitsky, 2009.
We need to make assumptions about size, directedness and bipartite-ness when we model and simulate the complete network.

- **Size**: Any size can be simulated, but if the model is fit using the observed frequencies, it should be used to simulate a population of that size unless a size adjustment is made in the simulation (see Krivitsky, Handcock and Morris 2011). We are going to work with a population size 500 here, equal to the number of respondents.

- **Directedness**: In this case the tie (“seen more than once”) it is undirected, so we will fit and simulate an undirected network. Note that ego data are in one sense inherently directed (respondents nominate alters, alters are not observed), but the type of tie that respondents report may be either directed or undirected, and that is what we are simulating.

- **Bipartite**: “Seen” is not a two-mode relationship, so we will fit and simulated a one-mode network. Note again that the ego data can be bipartite by design (egos vs. alters, if no alters are also respondents, or if data are collected on 2-mode networks) or not (if respondents are also alters). But again, the type of tie determines what we want to simulate.

In sum, we will simulate a one-mode, undirected network of size 500, assuming the ego statistics we observed contain the information we need to calculate the statistics that would have been observed in a self-contained population of that size, noting that other assumptions and approaches are possible.

**Initializing a network**

To ensure consistency between the degree distribution (which is a tabulation of nodes) and the mixing matrix (which is a cross-tabulation of ties) in our simulated “complete network”, it is important to recognize that in a complete network, the degree distribution frequencies should sum to twice the number of ties observed in the mixing matrix, because every tie is being reported by both nodes in the degree distribution. If we are fixing the population size at 500 in this simulation, then our observed mixing matrix data needs to be divided by 2.

Start by initializing an empty network of the desired size and assign the “sex” attribute to the nodes:

```r
go.net <- network.initialize(500, directed=F)
go.net %v% sex <- c(rep(0,250), rep(1,250))
```

Set up the observed statistics (adjusted for this “complete” network) as we will use them to assess the accuracy of the simulation later:

```r
go.deg <- c(180, 245, 60, 15) # node distn
go.mixmat <- matrix(c(164,44,26,176)/2, nrow=2, byrow=T) # adjusted tie distn
```

**Target statistics for the model**

Then, pick the observed statistics you want to target – we will start with a simple model here: the total number of ties (edges), and the number of sex-matched ties (homophily). These are both functions of the observed statistics:

```r
go.edges <- sum(go.mixmat)
go.sexmatch <- go.mixmat[1,1]+go.mixmat[2,2]
```

And combine these into a vector

```r
go.target.stats <- c(go.edges, go.sexmatch)
go.target.stats
```

```
[1] 205 170
```
ERGM model

Now, fit an ERGM to this network, with terms for the statistics you want to match, and the "target.stats" argument for `ergm` that specifies the target values for those statistics:

```r
ego.fit <- ergm(ego.net ~ edges + nodematch('sex'),
                target.stats = ego.target.stats)
```

Take a look at the fitted model:

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

```text
Summary of model fit

Formula: nw ~ edges + nodematch("sex")
<environment: 0x000000002ddac790>

Iterations: 8 out of 20

Monte Carlo MLE Results:

<table>
<thead>
<tr>
<th></th>
<th>Estimate</th>
<th>Std. Error</th>
<th>MCMC % p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>-7.4870</td>
<td>0.1690</td>
<td>0 &lt;1e-04 ***</td>
</tr>
<tr>
<td>nodematch.sex</td>
<td>1.5866</td>
<td>0.1857</td>
<td>0 &lt;1e-04 ***</td>
</tr>
</tbody>
</table>

---

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null Deviance: 172940 on 124750 degrees of freedom
Residual Deviance: 2941 on 124748 degrees of freedom

AIC: 2945 BIC: 2964 (Smaller is better.)
```

Simulation of a complete network from the model

Now that you have a fitted model, you can simulate a complete network from it, and look at the results:

```r
ego.sim1 <- simulate(ego.fit)
plot(ego.sim1, vertex.cex=.65, vertex.col="sex")
```

Diagnostics and model re-specification

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

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

```text
             degree0 degree1 degree2 degree3
sim   223     165     89     16
obs   180     245     60     15
```

```r
mixingmatrix(ego.sim1, "sex")
```
Figure 11: plot of chunk unnamed-chunk-43
Note: Marginal totals can be misleading for undirected mixing matrices.

0 1
0 80 40
1 40 90

ego.mixmat

      [,1] [,2]
[1,]   82  22
[2,]   13  88

We only targeted the number of same-sex ties, not the full mixing matrix.

The simulation stats seem quite different than the observed stats, and there are two possible reasons: either we mis-specified the original model (bias), or this one random draw may be unrepresentative of the distribution described by the model (variance). The latter is easily examined by simulating 100 networks, to see where the observed data fall in the distribution of networks produced by the model:

ego.sim100 <- simulate(ego.fit, nsim=100)
ego.sim100

Number of Networks: 100
Model: nw ~ edges + nodematch("sex")
Reference: ~Bernoulli
Constraints: ~.
Parameters:
                 edges  nodematch.sex
                 -7.487013  1.586633

More information can be obtained with

summary(ego.sim100)

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

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

                              edges nodematch.sex
                             sim 207.56       171.91
                             obs 205.00       170.00

These look pretty good – the means of the simulated target stats are within a few percent of the observed, depending on your computer configuration. We can plot the 100 replicates to see check the variation for any problematic patterns:

matplot(1:nrow(sim.stats), sim.stats,
pch=c("e","m","0","+"), cex=.65,
main="100 simulations from ego.fit model", sub="(default settings)",
 xlab="Replicate", ylab="frequency")
abline(h=ego.target.stats, col=c(1:4))
Figure 12: plot of chunk unnamed-chunk-47
The lines mark the target statistic frequencies in the observed data. The points represent the frequencies in the simulated networks.

The simulated network statistics vary stochastically around the target values, not trending over time.

The variation (about +/- 10%) represents the range of target statistics that are consistent with the fitted coefficients.

If you wanted instead to constrain these statistics to equal a specified value, then you can use the `constraints` argument during the `ergm` fit instead.

This is good for verifying that the simulation reproduces the target values we specified. So now let’s see whether the simulated complete networks also match statistics that were not set by the targets. We targeted edges and homophily.

How well does this model reproduce the full degree distribution?

```r
sim.fulldeg <- summary(ego.sim100 ~ degree(c(0:10)))
colnames(sim.fulldeg) <- paste("deg",0:10, sep='')
sim.fulldeg[1:5,]
```

<table>
<thead>
<tr>
<th>deg0</th>
<th>deg1</th>
<th>deg2</th>
<th>deg3</th>
<th>deg4</th>
<th>deg5</th>
<th>deg6</th>
<th>deg7</th>
<th>deg8</th>
<th>deg9</th>
<th>deg10</th>
</tr>
</thead>
<tbody>
<tr>
<td>242</td>
<td>167</td>
<td>75</td>
<td>10</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>220</td>
<td>187</td>
<td>71</td>
<td>15</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>223</td>
<td>183</td>
<td>73</td>
<td>15</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>216</td>
<td>183</td>
<td>82</td>
<td>14</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>204</td>
<td>196</td>
<td>76</td>
<td>18</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Recall that the degree range in our data was [0,3] by design, but we did not constrain the simulations to this range. If our model correctly captured the processes that led to the aggregate statistics we observe in our data, the simulated networks would show us what we missed. Here the simulated networks suggest that the fully observed network would have a wider range of degrees, which we might have observed, had we not truncated the data collection at 3 friends per respondent. About 1% of nodes have degree 4 or 5, and the max observed is 6.

But did our model did correctly capture the underlying processes? How well does the simulated degree distribution from this model match the frequencies we did observe? Aggregating the degrees of 3 or more in the simulations, we find:

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

<table>
<thead>
<tr>
<th>deg0</th>
<th>deg1</th>
<th>deg2</th>
<th>degree3+</th>
</tr>
</thead>
<tbody>
<tr>
<td>216.43</td>
<td>183.49</td>
<td>74.54</td>
<td>25.54</td>
</tr>
<tr>
<td>180.00</td>
<td>245.00</td>
<td>60.00</td>
<td>15.00</td>
</tr>
</tbody>
</table>

As with the single simulation above, the discrepancies between the simulation averages and the observed statistics are quite large. We can see this more clearly by plotting the degree frequencies for the 100 replicate networks we simulated:

```r
matplot(1:nrow(sim.deg), sim.deg, pch=as.character(0:3), cex=.5, main="Comparing ego.sims to non-targeted degree frequencies", sub = "(only total edges targeted)", xlab = "Replicate", ylab = "Frequencies")
abline(h=c(180, 245, 60, 15), col=c(1:4))
```

The simulations are producing systematically more isolates than expected (the “0” points vs. the black line), and systematically more degree 1 nodes. In fact, the two degree frequencies are essentially reversed in the simulation.

The fraction of nodes with 2 or 3+ partners is systematically off but by a much smaller amount.

So our observed network has fewer isolates than expected in a network of this density, more degree 1 nodes than expected, and fewer degree 2 and 3+ nodes.

This suggests the model is mis-specified. Since the degree 0 vs. degree 1 is the worst fitting aspect, we will try using the number of isolates as a target statistic in the model.
Figure 13: plot of chunk unnamed-chunk-50
ego.isolates <- ego.deg[1]
ego.target.stats <- c(ego.edges, ego.sexmatch, ego.isolates)
ego.fit <- ergm(ego.net ~ edges + nodematch('sex') + degree(0),
    target.stats = ego.target.stats)

Starting maximum likelihood estimation via MCMLE:
Iteration 1 of at most 20:
The log-likelihood improved by 0.0314
Step length converged once. Increasing MCMC sample size.
Iteration 2 of at most 20:
The log-likelihood improved by 0.004738
Step length converged twice. Stopping.

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

summary(ego.fit)

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

Formula: nw ~ edges + nodematch("sex") + degree(0)
<environment: 0x000000003168d2d8>

Iterations: 2 out of 20

Monte Carlo MLE Results:
                        Estimate Std. Error MCMC % p-value
edges           -8.4008    0.2427     0 <1e-04 ***
nodematch.sex   1.5892    0.1841     0 <1e-04 ***
degree0         -0.9606    0.1599     0 <1e-04 ***

---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null Deviance: 172940 on 124750 degrees of freedom
Residual Deviance: 2904 on 124747 degrees of freedom
AIC: 2910  BIC: 2939 (Smaller is better.)

Simulating from this model, the target statistics are again well matched:

ego.sim100 <- simulate(ego.fit, nsim=100,
    control=control.simulate.ergm(MCMC.interval=10000))
sim.stats <- attr(ego.sim100,"stats")
rbind(sim=colMeans(sim.stats), obs=ego.target.stats)

                  edges nodematch.sex degree0
sim    207.19  172.23  178.05
obs    205.00  170.00  180.00

And the full degree frequencies look much better:

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

```r
matplot(1:nrow(sim.deg), sim.deg, pch=as.character(0:3), cex=.5,
    main="Comparing ego.sims to non-targeted degree frequencies",
    sub = "(only 0, 2+ and total edges targeted)",
    xlab = "Replicate", ylab = "Frequencies")
abline(h=c(180, 245, 60, 15), col=c(1:4))
```

Figure 14: plot of chunk unnamed-chunk-54
The degree frequencies in these simulated networks are now well centered on the observed frequencies. So adding the one additional parameter to capture the lower than expected number of isolates did a good job of capturing how our observed network deviates from a random network with this density.

The fraction of nodes with 3+ partners produced by our new model might still be a bit low.

**Moral:** We can use ERGMs to estimate network models using target statistics from egocentrically sampled data. The fact that the target statistics are reproduced by this model does not guarantee that additional features of the network would also be reproduced. But starting with simple models can help to identify whether and how the aggregate statistics we observe from an egocentric sample deviate from those we would expect from the model. If we fit all of the observed statistics without a saturated model, we cannot reject the hypothesis that this model produced the network we sampled from.

We can also use this approach to explore network statistics that are not visible at all from the egocentric data, e.g., the geodesic distribution, betweenness, etc., but it must always be remembered that the distributions we will produce are based on our model. They faithfully reproduce the model, but that does not mean that the model faithfully represents the population.

In the STERGM workshop, we show how complete dynamic networks also can be simulated over time on the basis of egocentric data like these, with the minimal addition of a single estimate of partnership duration. For a movie of an example dynamic simulation used to explore the impact of network structure on HIV transmission, see http://statnet.org/movies

8. Additional functionality in the statnet family of packages

**Additional functionality**

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

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

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

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

**Statnet Commons: The development group**

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

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

**ergm**

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

**network**

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

References


