Introduction to Exponential-family Random Graph (ERG or p*)
modeling with *statnet*

INSNA Sunbelt – St. Pete Beach, Florida - Feb 2011

Carter T. Butts (University of California, Irvine)
Martina Morris (University of Washington)
Pavel N. Krivitsky (Penn State University)
Zack Almquist (University of California, Irvine)

Joint with the rest of the Statnet Development Team:
Mark S. Handcock (UCLA)
David R. Hunter (Penn State University)
Steven M. Goodreau (University of Washington)
Skye Bender de-Moll (Oakland)

Table of contents

> Two other workshops now cover this material

Basic resources

R webpage: http://www.r-project.org

Helpful **R** tutorials: http://cran.r-project.org/other-docs.html

statnet webpage: www.statnet.org

statnet help: statnet_help@statnet.org

Typographical conventions

Text in **`Courier bold`** represents code for you to type.

Text in `Courier regular` represents **R** output.

`# Text after pound signs is a comment`

All other text represents instructions and guidance.

SECTION 0. GETTING STARTED

Open an R session, and set your working directory to the location where you would like to save this work. You can do this with the pull-down menus (File>Change Dir) or with the command:

```
setwd('full.path.for.the.folder')
```

To install all of the packages in the statnet suite:
```
install.packages('statnet')
library(statnet)
```

Or, to only install the specific statnet packages needed for this tutorial:
```
install.packages('network')
install.packages('ergm')
install.packages('sna')
library(network)
library(ergm)
library(sna)
```

After the first time, to update the packages one can either repeat the commands above, or use:
```
update.packages('name.of.package')
```

For this tutorial, we will need one additional package (**coda**), which is recommended (but not required) by **ergm**:
```
install.packages('coda')
library(coda)
```

SECTION 1. STATISTICAL NETWORK MODELING; THE *ERGM* COMMAND AND *ERGM* OBJECT.

Make sure the statnet package is attached:

```
library(statnet)
```

or

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

The ergm package contains several network data sets that you can use for practice examples.

```
data(package='ergm')          # tells us the datasets in our packages
data(florentine)              # loads flomarriage & flobusiness data
flomarriage                   # Let's look at the flomarriage data
plot(flomarriage)             # Let's view the flomarriage network
```

Remember the general ergm representation of the probability of the observed network, and the conditional log-odds of a tie:

$P(Y=y) = \exp[\theta'g(y)] / k(\theta)$	# Y is a network, $g(y)$ is a vector of network stats # θ is the vector of coefficients, $k(\theta)$ is a normalizing constant
$\text{logit}(P(Y_{ij}=1 \mid Y^c)) = \theta'\Delta(g(y))_{ij}$	# Y_{ij} is an actor pair in Y, Y^c is the rest of the network, # $\Delta(g(y))_{ij}$ is the change in $g(y)$ when the value of # Y_{ij} is toggled on

We begin with the simplest possible model, the Bernoulli or Erdös-Rényi model, which contains only an edge term.

```
flomodel.01 <- ergm(flomarriage~edges)                    # fit model
flomodel.01                                               # look at the model

Newton-Raphson iterations:  5

MLE Coefficients:
 edges
-1.609
```

summary(flomodel.01)   # look in more depth

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


Formula:    flomarriage ~ edges

Newton-Raphson iterations:  5

Maximum Likelihood Results:
       Estimate Std. Error MCMC s.e. p-value
edges   -1.6094     0.2449        NA  <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

For this model, the pseudolikelihood is the same as the likelihood.

    Null  Deviance: 166.355  on 120  degrees of freedom
 Residual Deviance: 108.135  on 119  degrees of freedom
          Deviance:  58.221  on   1  degrees of freedom

AIC: 110.13    BIC: 112.92
```

How to interpret this model? The log-odds of any tie occurring is:

    = -1.609 * change in the number of ties

    = -1.609 * 1   # for all ties, since the addition of *any* tie to the
                                          # network changes the number of ties by 1!

Corresponding probability is:

    = exp(-1.609) / (1+ exp(-1.609))

    = 0.1667   # what you would expect, since there are 20/120 ties

Let's add a term often thought to be a measure of "clustering" -- the number of completed triangles

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

```
# Note we're in stochastic simulation now – your output will differ

flomodel.02 <- ergm(flomarriage~edges+triangle)
Iteration 1 of at most 20:
the log-likelihood improved by 0.001786
Iteration 2 of at most 20:
the log-likelihood improved by 0.0005837
Iteration 3 of at most 20:
the log-likelihood improved by 0.0001311
Iteration 4 of at most 20:
the log-likelihood improved by < 0.0001
Iteration 5 of at most 20:
Convergence detected. Stopping early.
```

```
the log-likelihood improved by < 0.0001
```

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

summary(flomodel.02)

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

Formula:   flomarriage ~ edges + triangle

Iterations:  20

Monte Carlo MLE Results:
        Estimate Std. Error MCMC % p-value
edges     -1.6748     0.3518      0  <1e-04 ***
triangle   0.1557     0.5960      0   0.794
---
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 `flomodel.02` run
  > flomodel.02<-logLik(flomodel.02, add=TRUE)
to add it to the object or rerun this function with eval.loglik=TRUE.
```

Again, how to interpret coefficients?

Conditional log-odds of two actors forming a tie is:

-1.673 * change in the number of ties + 0.139 * change in number of triangles

if the tie will not add any triangles to the network, its log-odds. is -1.673.
if it will add one triangle to the network, its log-odds is -1.673 + 0.139 = -1.534
if it will add two triangles to the network, its log-odds is: -1.673 + 0.139*2 = -1.395

the corresponding probabilities are 0.158, 0.177, and 0.199.

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

class(flomodel.02)   # this has the class ergm
```
[1] 'ergm'
```

names(flomodel.02)   # let's look straight at the ERGM obj.

```
 [1] "coef"          "sample"        "sample.obs"    "iterations"    "MCMCtheta"
 [6] "loglikelihood" "gradient"      "covar"         "failure"       "mc.se"
[11] "network"       "newnetwork"    "coef.init"     "initialfit"    "coef.hist"
[16] "stats.hist"    "null.deviance" "etamap"        "formula"       "target.stats"
[21] "constrained"   "constraints"   "control"       "reference"     "estimate"
[26] "offset"        "drop"          "estimable"     "mle.lik"
```

flomodel.02$coef   # the $ allows you to pull an element out from
flomodel.02$formula   # a list
flomodel.02$mle.lik

```
wealth <- flomarriage %v% 'wealth'      # the %v% extracts vertex attributes from a
wealth                                  # network
plot(flomarriage, vertex.cex=wealth/25) # network plot with vertex size
                                        # proportional to wealth
```

We can test whether edge probabilities are a function of wealth:

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

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

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

Newton-Raphson iterations: 4

Maximum Likelihood Results:
                Estimate Std. Error MCMC s.e. p-value
edges           -2.594929   0.536056        NA  <1e-04 ***
nodecov.wealth   0.010546   0.004674        NA  0.0259 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

For this model, the pseudolikelihood is the same as the likelihood.

    Null  Deviance: 166.355  on 120  degrees of freedom
 Residual Deviance: 103.109  on 118  degrees of freedom
          Deviance:  63.247  on   2  degrees of freedom

AIC: 107.11    BIC: 112.68
```

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

****

```
data(samplk)                            # Let's try a model or two on
ls()                                    # directed data: Sampson's Monks
samplk3
plot(samplk3)
sampmodel.01 <- ergm(samplk3~edges+mutual)# Is there a statistically significant
summary(sampmodel.01)                   # tendency for ties to be reciprocated
                                        # ("mutuality")?
data(faux.mesa.high)                    # Let's try a larger network
mesa <- faux.mesa.high
plot(mesa)
mesa
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))

summary(fauxmodel.01)
```

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

```
table(mesa %v% "Race")                          # Frequencies of race
mixingmatrix(mesa, "Race")
```

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

Time to consider some missing data:

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

The coefficient equals -2.590. This is the logodds of the probability .0698. Our network has 3 ties, out of the 43 nodal pairs (10 choose 2 minus 2) whose dyad status we have observed. 3/43 = 0.0698.

```
ergm(missnet~edges+degree(2))
```

```
missnet[4,6] <- missnet[4,9] <- 0
```

```
ergm(missnet~edges+degree(2))
```

The two estimates for the degree2 coefficient differ considerably. In the first case, there is one node we know for sure has degree 2, two that may or may not, and seven that we know for sure do not. In the latter, there is one node that has degree 2, and nine that do not.

SECTION 2. 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 **ergm** (to estimate an ergm model)
> calls to **simulate** (to simulate networks from an ergm model fit)
> calls to **summary** (to obtain measurements of network statistics on a dataset)

4.1. Terms provided with ergm

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

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

For a more complete discussion of these terms see the 'Specifications' paper in *J Stat Software* v. 24. (link is available online at www.statnet.org)

4.2. Coding new terms

We have recently released a new package (**ergm.userterms**) and tutorial aimed at making it much easier than before to write one's own terms. The package is available on CRAN, and installing it will also download the tutorial (ergmuserterms.pdf). We teach a workshop at the Sunbelt meetings, and are also hoping for the tutorial to appear soon in the *Journal of Statistical Software*. 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).

SECTION 3. NETWORK SIMULATION: THE *SIMULATE* COMMAND AND *NETWORK.LIST* OBJECTS.

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

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

```
class(flomodel.03.sim)
[1] 'network.list'
```

```
summary(flomodel.03.sim)
Number of Networks: 10
Model: flomarriage ~ edges + nodecov("wealth")
Reference: Bernoulli
Constraints: ~.
Parameters:
        edges nodecov.wealth
   -2.59492903     0.01054591
```

```
Stored network statistics:
      edges nodecov.wealth
 [1,]    20           2089
 [2,]    25           2432
 [3,]    21           1897
 [4,]    27           2956
 [5,]    18           2094
 [6,]    24           2761
 [7,]    22           1926
 [8,]    14           1551
 [9,]    19           1857
[10,]    21           1878
```

```
length(flomodel.03.sim)
[1] 10
```

```
flomodel.03.sim[[1]]                   # double brackets pull an element
                                       # out of a list by position #
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
```

```
plot(flomodel.03.sim[[1]], label= flomodel.03.sim[[1]] %v% "vertex.names")
```
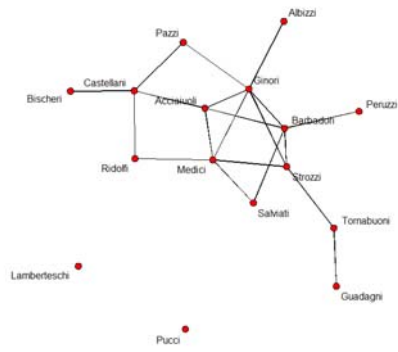
Voilà. (Of course, yours will look somewhat different.)

**SECTION 4. EXAMINING THE QUALITY OF MODEL FIT – *GOF*.**

ERGMs are generative models – that is, they represent the process that governs tie formation at a local level. These local 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 model "fits the data" is therefore how well it reproduces these global properties. 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.

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

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


Goodness-of-fit diagnostics
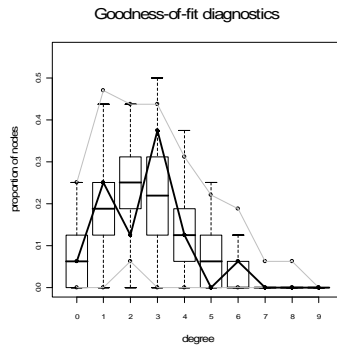
```
mesamodel.02 <- ergm(mesa~edges)
mesamodel.02.gof <- gof(mesamodel.02~distance,nsim=10)
plot(mesamodel.02.gof)
```


Goodness-of-fit diagnostics
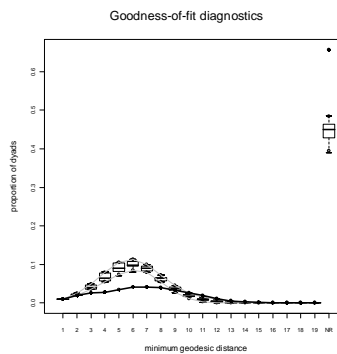
(for a good example of model exploration and fitting for the Add Health Friendship networks, see Goodreau Kitts & Morris **Demography** 2009)

SECTION 5. DIAGNOSTICS: TROUBLESHOOTING AND CHECKING FOR MODEL DEGENERACY

The computational algorithms in **ergm** use MCMC to estimate the likelihood function. Part of this process involves simulating a set of networks to approximate unknown components of the likelihood.

When a model is not a good representation of the observed network the estimation process may be affected. In the worst case scenario, the simulated networks will be so different from the observed network that the algorithm fails altogether. This can occur for two general reasons. First, the simulation algorithm may fail to converge, and the sampled networks are thus not from the specified distribution. Second, the model parameters used to simulate the networks are too different from the MLE, so even though the simulation algorithm is producing a representative sample of networks, this is not the sample that would be produced under the MLE.

For more detailed discussions of model degeneracy in the ERGM context, see the papers in *J Stat Software* v. 24. (link is available online at <u>www.statnet.org</u>)

We can use diagnostics to see what is happening with the simulation algorithm, and these can lead us to ways to improve it. We will first consider a simulation where the algorithm works. To understand the algorithm, consider

```
fit <- ergm(flobusiness~edges+degree(1),
      control=control.ergm(MCMC.interval=1, MCMC.burnin=1000, seed=1))
```

This runs a version with every network returned. Let us look at the diagnostics produced:

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

Let's look more carefully at a default model fit:

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

To see the diagnostics use:

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

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

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

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

Very interesting. You could have gotten some more feedback about this during the fitting, by using:

```
fit <- ergm(magnolia~edges+triangle,
 control=control.ergm(seed=1),
 verbose=T)
```

You might try to increase the MCMC sample size:

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

Now, try it again with a sample size of 50,000.

How about trying the more robust version of modeling triangles -- GWESP? (For a technical introduction to GWESP see Hunter and Handcock; for a more intuitive description and empirical application see Goodreau Kitts and Morris 2009)

```
fit <- ergm(magnolia~edges+gwesp(0.5,fixed=T),
 control =  control.ergm(seed=1))
mcmc.diagnostics(fit)
```

Still degenerate, but maybe getting closer?

```
fit <- ergm(magnolia~edges+gwesp(0.5,fixed=T)+nodematch('Grade')+nodematch('Race')+
      nodematch('Sex'),
 control = control.ergm(seed=1),
 verbose=T)
```

```
pdf('diagnostics1.pdf') #Use the recording function if possible, otherwise send to
pdf
mcmc.diagnostics(fit)
dev.off()               #If you saved to pdf, look at the file
```

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

```
pdf('diagnostics2.pdf') #Ditto
mcmc.diagnostics(fit)
dev.off()               #If you saved to pdf, look at the file
```

```
args(ergm)
```

```
fit <- ergm(magnolia~edges+gwesp(0.25,fixed=T)+nodematch('Grade')+nodematch('Race')+
      nodematch('Sex'),
 control = control.ergm(seed=1,MCMC.samplesize=50000,MCMC.interval=1000),
 verbose=T)
pdf('diagnostics3.pdf') #Ditto
mcmc.diagnostics(fit)
dev.off()               #If you saved to pdf, look at the file
```

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

SECTION 6. WORKING WITH EGOCENTRICALLY SAMPLED NETWORK 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:

Degee distribution (number of friends seen > once)

Degree	Frequency	Fraction	Ties
0	180	0.36	0
1	245	0.49	245
2	60	0.12	120
3	15	0.03	45
Total	500	1.00	410

Sex Mixing Matrix

		Friend's sex		
		M	*F*	*Total*
Respondent	*M*	164	44	
Sex	*F*	26	176	
	Total			410

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: Ego data are in one sense inherently directed (respondents nominate alters, alters are not observed), but the *relationship* may be either. In this case ("seen more than once") it is undirected, so we will fit and simulate an undirected network.
- Bipartite: Ego data can be bipartite (if no alters are also respondents, or data are collected on 2-mode networks) or not (if respondents are also alters). But again, the relationship may be either. "Seen" is undirected, and we will fit and simulated and undirected network.

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 are possible.

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 should imply *twice* the number of ties observed in the mixing matrix, because every tie is being reported by both

nodes in the degree distribution. If we are fixing the population size at 500 in this simulation, then our observed mixing matrix data needs to be divided by 2.

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

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

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

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

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

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

And combine these into a vector

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

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

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

Iteration 1 of at most 20:
Convergence detected. Stopping early.
the log-likelihood improved by 0.0001152

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")
<environment: 0x0f83a7ac>

Iterations:  20

Monte Carlo MLE Results:
            Estimate Std. Error MCMC % p-value
edges         -7.4889     0.1689      0  <1e-04 ***
nodematch.sex  1.5877     0.1822      0  <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

    Null  Deviance: 172940.2  on 124750  degrees of freedom
```

```
  Residual Deviance:   2940.7  on 124748  degrees of freedom
          Deviance: 169999.5  on      2  degrees of freedom


AIC: 2944.7     BIC: 2964.2
```

Take a look at the fitted model:

summary(ego.fit)

```
summary(ego.fit)

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


Formula:    nw ~ edges + nodematch("sex")


Iterations:   20


Monte Carlo MLE Results:
               Estimate Std. Error MCMC % p-value
edges           -7.4889     0.1689      0  <1e-04 ***
nodematch.sex    1.5877     0.1822      0  <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null  Deviance: 172940.2  on 124750  degrees of freedom
 Residual Deviance:   2940.7  on 124748  degrees of freedom
          Deviance: 169999.5  on      2  degrees of freedom


AIC: 2944.7     BIC: 2964.2
```
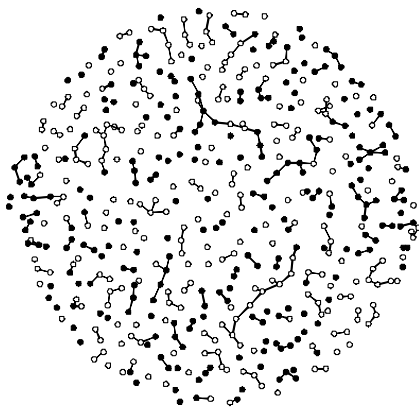
Now that you have a fitted model, you can simulate a complete network from it, and look at the results:

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

Does it reproduce the observed degree and mixing frequencies?

rbind(summary(ego.sim1 ~ degree(c(0:3))), ego.deg)

```
        degree0 degree1 degree2 degree3
            204     199      71      20
ego.deg     180     245      60      15
```
mixingmatrix(ego.sim1, "sex")

> We only targeted the total number of edges, not the full degree distribution

```
Note:  Marginal totals can be misleading
 for undirected mixing matrices.
    0   1
0  95  37
1  37  81
> ego.mixmat
      [,1] [,2]
[1,]   82   22
[2,]   13   88
```

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")
Constraints: ~.
Parameters:
        edges nodematch.sex
    -7.488886     1.587689
```

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(colMeans(sim.stats), ego.target.stats)

```
                   edges nodematch.sex
                  203.06           169
ego.target.stats 205.00           170
```

These look pretty good – the means of the simulated target stats are within 1% of the observed. 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))

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

100 simulations from ego.fit model
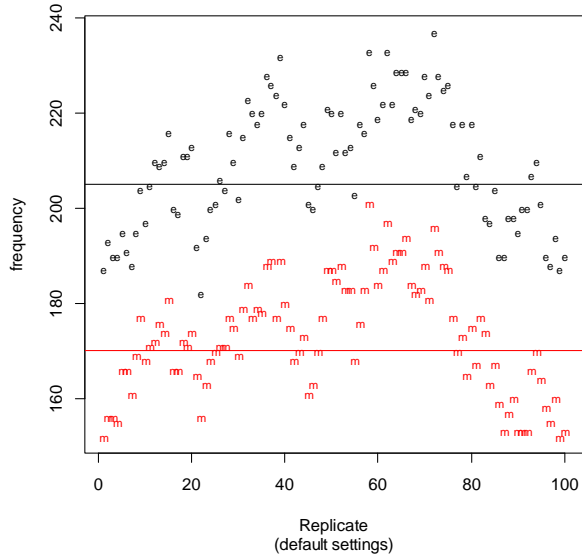
The lines mark the target statistic frequencies in the observed data. The points represent the frequencies in the simulated networks.

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

But – there is clear autocorrelation across the replicates, which suggests we might want to increase the MCMC interval to draw more independent realizations.

```
ego.sim100 <- simulate(ego.fit, nsim=100,
  control=control.simulate.ergm(MCMC.interval=10000))
```


100 simulations from ego.fit model
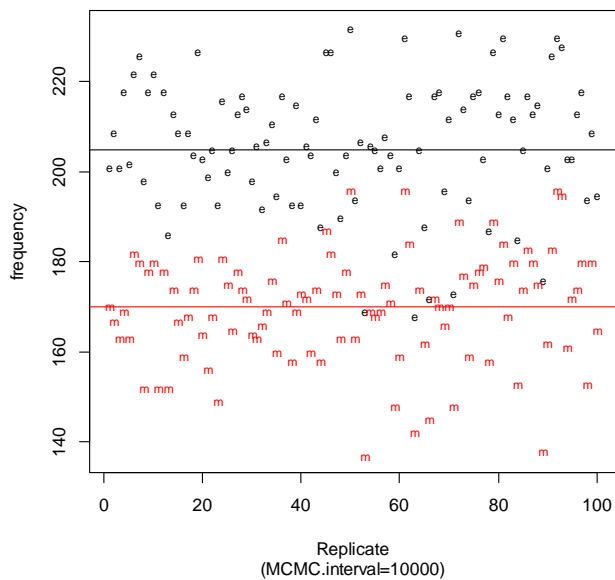
With the larger interval, the autocorrelation is no longer detectable, and all of the statistics from the simulated networks vary in a symmetric band around their targets.

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

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

This is good for verifying that the simulation reproduces the target values we specified. So now let's see whether the simulated complete networks also match statistics that were not set by the targets. We targeted edges and homophily. How well does this model reproduce the full degree distribution?

```
sim.fulldeg <- summary(ego.sim100 ~ degree(c(0:10)))
sim.fulldeg
     degree0 degree1 degree2 degree3 degree4 degree5 degree6 degree7 degree8 degree9 degree10
[1,]    198     190      81      26       5       0       0       0       0       0       0
[2,]    207     177      81      33       2       0       0       0       0       0       0
[3,]    201     178      87      29       4       1       0       0       0       0       0
```

```
        [4,]     207     173      86      27       7       0       0       0       0       0       0
        [5,]     219     168      80      26       7       0       0       0       0       0       0
        …
```

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

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

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

          degree0 degree1 degree2 degree3+
           223.97  176.47   74.87    24.69
ego.deg    180.00  245.00   60.00    15.00
```

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

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

Comparing ego.sims to non-targeted degree frequencies
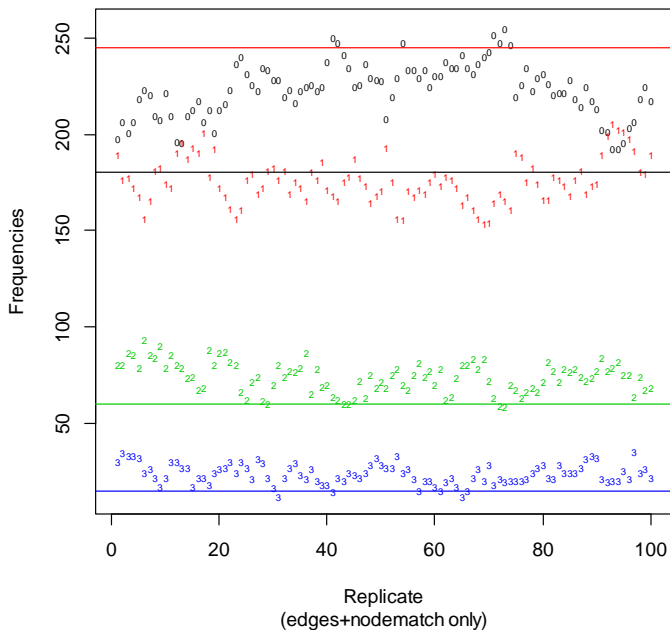
The simulations are producing systematically more isolates than expected (the "0" points vs. the black line), and systematically more degree 1 nodes. In fact, the two degree frequencies are essentially reversed in the simulation.

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

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

This suggests the model is mis-specified. Since the degree 0 vs. degree 1 is the worst fitting aspect, we will try using the number of isolates as a target statistic in the model.

```
ego.isolates <- ego.deg[1]
ego.target.stats <- c(ego.edges, ego.sexmatch, ego.isolates)
ego.fit <- ergm(ego.net ~ edges + nodematch('sex') + degree(0),
 target.stats = ego.target.stats)
summary(ego.fit)

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

Formula:   nw ~ edges + nodematch("sex") + degree(0)

Iterations:  20

Monte Carlo MLE Results:
              Estimate Std. Error MCMC % p-value
edges          -8.4012     0.2462      1  <1e-04 ***
nodematch.sex   1.5821     0.1854      1  <1e-04 ***
degree0        -0.9601     0.1590      0  <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

    Null  Deviance: 172940.2  on 124750  degrees of freedom
 Residual Deviance:   2902.9  on 124747  degrees of freedom
          Deviance: 170037.3  on       3  degrees of freedom

AIC: 2908.9    BIC: 2938.1
```

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(colMeans(sim.stats), ego.target.stats)
                  edges nodematch.sex degree0
                 205.61        169.99  178.79
ego.target.stats 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(colMeans(sim.deg),ego.deg)
        degree0 degree1 degree2 degree3+
         178.79   245.2   63.65    12.36
ego.deg  180.00   245.0   60.00    15.00
```

and finally, plotting the full degree frequencies

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


Comparing ego.sims to non-targeted degree frequencies

Replicate
(edges + nodematch + isolates)
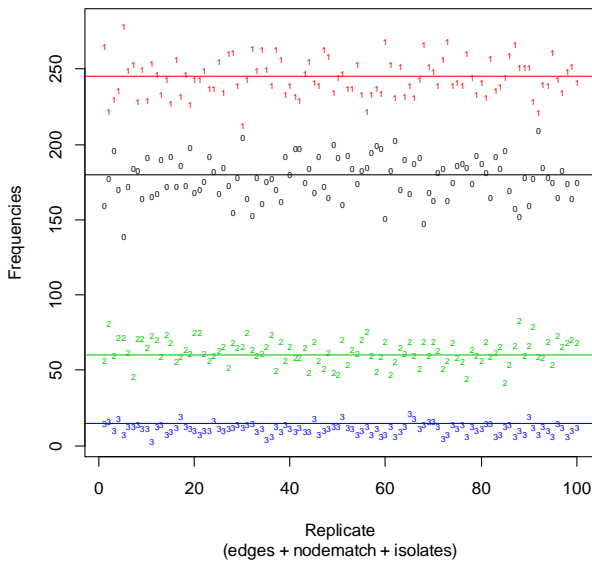
The degree frequencies in these simulated networks are now well centered on the observed frequencies. So adding the one additional parameter to capture the lower than expected number of isolates did a good job of capturing how our observed network deviates from a random network with this density.

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 statnet.org/movies.

SECTION 8. ADDITIONAL FUNCTIONALITY

8.1. Additional functionality

The **statnet** suite of packages currently contains many additional features not covered in this tutorial:

- tools for fitting dynamic network models (**stergm,** in the **ergm** base package)
- tools for fitting relational event models (**relevent** package)
- curved exponential family estimation and simulation (**ergm** base package)
- latent space and latent cluster analysis (**latentnet** package)
- network permutation models (**netperm** package)
- MLE estimation for degree distributions (negative binomial, Poisson, scale-free, etc.) (**degreenet** package)
- analysis of bipartite networks (**network** package)
- simulation of bipartite networks with given degree distributions (**networksis** package)
- hierarchical ERGMs (**hergm** package)

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

8.2. Additional functionality in development:

- ERGMs for valued ties – expected later 2012
- Temporal ERGMs (TERGMs) for longitudinal network panel data, and other temporal extensions – expected later 2012
- Temporally extended (vertex and edge) attributes for TERGMS – expected later 2012
- Network movie maker: ndTV – functionality previewed in STERGM workshop this year, CRAN release expected later 2012

8.3. 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 <krivitsky@stat.psu.edu>
```

References:

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

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).

Krivitsky, P.N. Statistical Models for Social Network Data and Processes. PhD thesis, University of Washington, Seattle, WA, August 2009.

Krivitsky, P. N., M. S. Handcock and M. Morris (2011). "Network Size and Composition Effects in Exponential-Family Random Graph Models." Statistical Methodology **forthcoming**.

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.

Appendix B: Table of existing statnet terms

(see also: Morris, M., D. Hunter and M. Handcock (2008). "Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects." Journal of Statistical Software **42**(i04).

Term	Undir?	Dir?	Bip?	Required Args	Optional Args
Basic terms					
edges	X	X	X		
density	X	X	X		
mutual		X			
asymmetric		X			
meandeg	X	X	X		
Nodal attribute terms					
nodecov (aka nodemain)	X	X	X	attrname	
nodefactor	X	X	X	attrname	
nodeifactor		X		attrname	
nodeofactor		X		attrname	
nodeicov		X		attrname	
nodeocov		X		attrname	

nodemix	X	X	X	attrname	
nodematch (aka match)	X	X	X	attrname	
absdiff	X	X	X	attrname	
absdiffcat	X	X	X	attrname	
b1factor			X	attrname	base
b2factor			X	attrname	base
smalldiff	X	X	X	attrname, cutoff	
Relational attribute terms					
edgecov	X	X	X	network or attrname	
dyadcov	X	X	X	network or attrname	
hamming	X	X	X	network or attrname	
hammingmix		X		network or attrname	base
Degree terms					
degree	X			vec. of degrees	by
idegree		X		vec. of degrees	by
odegree		X		vec. of degrees	by
b1degree			X	vec. of degrees	by
b2degree			X	vec. of degrees	by
gwdegree	X			decay	fixed
gwidegree		X		decay	fixed
gwodegree		X		decay	fixed
gwb1degree			X	decay	fixed
gwb2degree			X	decay	fixed
isolates	X	X	X		
concurrent	X				by
b1concurrent			X		by
b2concurrent			X		by
degcor	X				
degcrossprod	X				
adegcor	X				
rdegcor	X				
indegreepopularity		X			
outdegreepopularity		X			
Star terms					
kstar	X		X	vec. of star sizes	attrname
istar		X		vec. of star sizes	attrname
ostar		X		vec. of star sizes	attrname
b1star			X	vec. of star sizes	attrname
b2star			X	vec. of star sizes	attrname
b1twostar			X	b1attrname, b2attrname	base
b2twostar			X	b1attrname, b2attrname	base
b1starmix			X	k, attrname	base, diff
b2starmix			X	k, attrname	base, diff
m2star		X			
altkstar	X	X	X	lambda	fixed
Cycle and triangle terms					
triangle (aka triangles)	X	X			attrname
ctriple (aka ctriad)		X			attrname
ttriple (aka ttriad)		X			attrname
tripercent	X	X			attrname
cycle	X	X		vec. of cycle sizes	
localtriangle	X	X		network or attrname	
balance	X	X			

triadcensus	X	X		triad types to include	
intransitive		X			
nearsimmelian		X			
simmelian		X			
simmelianties		X			
transitive		X			
transitiveties		X			attrname
Actor-specific effects					
receiver		X			base
sender		X			base
sociality	X				attrname, base
Shared partner terms					
esp (edgewise shared ptnrs)	X	X		vec. of partner #s	
dsp (dyadwise shared ptnrs)	X	X	X	vec. of partner #s	
nsp (nonedge shared ptnrs)	X	X		vec. of partner #s	
gwesp	X	X		alpha	fixed
gwdsp	X	X	X	alpha	fixed
gwnsp	X	X		alpha	fixed
Paths					
towpath	X	X	X		
threepath	X	X	X		