Modeling Networks with Missing Data in *statnet*

Political Networks Conference, Ann Arbor, MI - June 2011

Carter T. Butts (University of California, Irvine)
Zack W. Almquist (University of California, Irvine)
Sean Fitzhugh (University of California, Irvine)
Lorien Jasny (University of California, Irvine)
Nicole Pierski (University of California, Irvine)
Emma Spiro (University of California, Irvine)

Joint with the rest of the Statnet Development Team:
Steven M. Goodreau (University of Washington)
Mark S. Handcock (UCLA)
David R. Hunter (Penn State University)
Pavel N. Krivitsky (Carnegie Mellon University)
Martina Morris (University of Washington)

Table of contents

Basic resources

Workshop webpage: http://polnet2011.statnet.org/
R webpage: http://www.r-project.org
Helpful **R** tutorials: http://cran.r-project.org/other-docs.html
statnet webpage: http://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 an additional data file (**`polnet2011.Rdata`**), which can be obtained from the workshop web site (http://polnet2011.statnet.org/). Be sure to put this file in your working directory, so that you can load it at need.

For today's workshop, you will need the latest version of **`ergm`**. After loading the **`ergm`** library, you can verify the current version by typing

```
sessionInfo()
```

You should find that **`ergm`** is version 2.4-1 or later. If not, you'll need to get the latest version, either from the workshop web page or from one of us.

SECTION 1: A QUICK INTRODUCTION TO `network`: DATA IMPORT, EXPLORATION, MANIPULATION

1.1 Built-in Datasets

```
library(network)                          #Make sure that network is loaded
data(package='network')                   #List available datasets in network
data(flo)                                 #Load a built-in data set; see ?flo for more
flo                                       #Examine the flo adjacency matrix
?flo                                      #More information about these data
```

1.2 Creating network Objects

```
class(flo)                                #Class of original object
nflo <- network(flo, directed=FALSE)      #Create a network object based on flo
class(nflo)                               #Class of new object
nflo                                      #Get a quick description of the data
nempty <- network.initialize(5)           #Create an empty graph with 5 vertices
nempty
```

1.3 Description and Visualization

```
summary(nflo)                             #Get an overall summary
network.size(nflo)                        #How large is the network?
network.edgecount(nflo)                   #How many edges are present?
plot(nflo,displaylabels=T,boxed.labels=F) #Plot with names
```

1.4 Adding, Deleting, Testing Edges

```
g <- network.initialize(5)                #Create an empty graph
g[1,2] <- 1                               #Add an edge from 1 to 2
g                                         #Examine the result
g[1,2] <- 0                               #Remove the edge from 1 to 2
g                                         #It's gone!
```

1.5 Testing Adjacency

```
nflo[9,3]                                 #Medici to Barbadori?
nflo[9,]                                  #Entire Medici row
nflo[1:4,5:8]                             #Subsets are possible
nflo[-9,-9]                               #Negative numbers _exclude_ nodes
```

1.6 Add vertex attributes

```
nflo
nflo %v% 'woo' <- letters[1:16]           # Same as set.vertex.attribute
nflo %v% 'woo'                            # Same as get.vertex.attribute
```

1.7 Sample **sna** *Routines*

```
library(sna)                              #Load the sna library
betweenness(flo)
isolates(nflo)
kcycle.census(nflo,mode='graph',maxlen=5)
```

SECTION 2. FITTING A BASIC ERG MODEL; 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
$\text{logit}(P(Y_{ij}=1)) = \theta' \Delta(g(y))_{ij}$	# Y_{ij} is an actor pair in Y, $\Delta(g(y))_{ij}$ 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 prob. = 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.

```
set.seed(1664)          #Seeding the RNG for demo purposes; otherwise, your output will vary
flomodel.02 <- ergm(flomarriage~edges+triangle)

Iteration 1 of at most 3:
the log-likelihood improved by 0.001723

Iteration 2 of at most 3:
the log-likelihood improved by 0.01581

Iteration 3 of at most 3:
the log-likelihood improved by 0.1805

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

MCMC sample of size 10000

Monte Carlo MLE Results:
         Estimate Std. Error MCMC s.e. p-value
edges     -1.6913     0.3284     0.038  <1e-04 ***
triangle   0.4076     0.3080     0.012   0.188
---
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 the coefficients?

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

-1.691 * change in the number of ties + 0.408 * change in number of triangles

if the tie will not add any triangles to the network, its log-odds. is -1.691.
if it will add one triangle to the network, its log-odds is -1.691 + 0.408 = -1.283
if it will add two triangles to the network, its log-odds is: -1.691 + 0.408*2 = -0.875

the corresponding probabilities are 0.156, 0.217, and 0.294.

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.miss"    "iterations"
 [5] "MCMCtheta"      "loglikelihood"  "gradient"       "covar"
 [9] "samplesize"     "failure"        "mc.se"          "burnin"
[13] "interval"       "network"        "newnetwork"     "theta.original"
[17] "mplefit"        "parallel"       "null.deviance"  "etamap"
[21] "formula"        "constraints"    "prop.weights"   "offset"
[25] "drop"
```

By default, ergm does not estimate the log-likelihood for models fit using MCMC; that is because this calculation involves the use of additional simulation, and can be time consuming. In order to obtain the log-likelihood, we use the `logLik` function. (It only has to be done once per object - thereafter, the likelihood information is stored in the model object.)

```
flomodel.02 <- logLik(flomodel.02, add=TRUE)    # Compute the log-likelihood, and update
summary(flomodel.02)                            # Voila!  Fit information is now present!
```

Model elements can also be selected directly:

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

We've seen how to add a dependence term to a model. Now, let's consider covariate effects. In the case of the Florentine families, marriage ties were part of a larger struggle over power and wealth. Could family wealth be related to the marriage network?

```
wealth <- flomarriage %v% 'wealth'      # the %v% extracts vertex attributes from a network
wealth
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 association between familial wealth and the probability of a tie. (Note that here the probability of a tie conditionally related to the total wealth of the families involved. You can also model the probability of a tie as a function of wealth *differences*, or more exotic things. See `?"ergm-terms"` for more information!)

****

Now, let's try a model or two on directed data. We begin with attributions of "liking" among Sampson's Monks:

```
data(samplk)                                    # Load the data set
ls()                                            # See the object
samplk3
plot(samplk3)
sampmodel.01 <- ergm(samplk3~edges+mutual)      # Is there a statistically significant tendency
summary(sampmodel.01)                           # for ties to be reciprocated ("mutuality")?
```

Although the cases we have seen so far are fairly small, we can work with larger cases. Here's another:

```
data(faux.mesa.high)                            # Not an actual school - but it looks like one!
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. Technically, this means that the MLE here does not exist: ever more negative coefficients would give us ever higher likelihoods (albeit with diminishing marginal returns), and no finite value is a maximum. In such situations, `ergm` inserts the limit to which the optimization process approaches.

SECTION 3. NETWORK TERMS AVAILABLE FOR *ERGM* MODELING and SIMULATION

Terms are the expressions (e.g. “triangle”) used on the right-hand size of :

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

3.1. Terms provided with statnet

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)

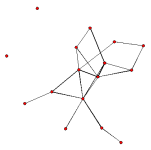
3.2. Coding new terms

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 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 4. NETWORK SIMULATION: THE *simulate* COMMAND AND *NETWORK.SERIES* OBJECTS.

Once we have estimated the coefficients of an ERGM, the model is completely specified. It defines a probability distribution across all networks satisfying a user-specified set of constraints (by default, size and directedness). 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.series'

names(flomodel.03.sim)
[1] 'formula'  'networks' 'stats'    'coef'

length(flomodel.03.sim$networks)
[1] 10

flomodel.03.sim$networks[[1]]            # double brackets pulls an element
                                         # out of a list by position #
Network attributes:
   vertices = 16
   directed = FALSE
   hyper = FALSE
   loops = FALSE
   multiple = FALSE
   bipartite = FALSE
   total edges= 24
     missing edges= 0
     non-missing edges= 20

Vertex attribute names:
    priorates totalties vertex.names wealth

plot(flomodel.03.sim$networks[[1]])
```



Voilà. (Of course, yours will look somewhat different.)

SECTION 5. MODELING WITH CENSORING AND OTHER CONSTRAINTS.

A standard model in ERGM form has three components: a vector of statistics (the "terms" in the model), a vector of parameters, and a support constraint. This last indicates the set of networks over which the probability distribution is defined - i.e., the set of potentially observable networks. By default, `ergm` takes the support to be equal to all graphs having the same size and directedness as the graph to which the model is fit. Thus, if you fit an ERGM to a 17-node directed network, the `ergm` function will assume that the set of possible networks you *could* have observed is the set of all 17-node directed graphs.

Usually, this is fine, but sometimes it's not good enough: we may wish to place additional constraints on the support of the model. The most common reason for doing so is to capture features of the measurement process (e.g., censoring). For instance, a network may be measured by collecting exactly one outgoing tie from each individual (a common design in very old sociometric surveys). When modeling this network, we must take into account the fact that each node has an outdegree of exactly 1 (no more, and no less!); otherwise, the model will try to "account for" this degree distribution as an endogenous phenomenon, possibly producing misleading results.

To see an example of this, let's return to Sampson's Monastery data:

```
sampmodel.01 <- ergm(samplk3~edges+mutual)      # Fitting a basic, unconstrained model
```

How well does the model reproduce the features of the data? To assess, we use the "goodness-of-fit" (or `gof`) function. `gof` will simulate a set of networks from the fitted model, examine a variety of structural characteristics for those networks, and compare them to the same characteristics on the original data. If the model is working well, then we expect features of interest in the observed network to be within the range of typical outcomes produced by the model. (This is sometimes called an "adequacy check.")

```
sampmodel.01.gof <- gof(sampmodel.01)          # Run the goodness-of-fit routine
par(mfrow=c(2,2))                              # Plot the results as a 2x2 display
plot(sampmodel.01.gof)
par(mfrow=c(1,1))
```

There's something very strange about this case -- note the poorly modeled outdegree distribution. Naively, we might try to model this using degree distribution effects (e.g., `gwodegree`). Let's try that:

```
sampmodel.02 <- ergm(samplk3~edges+mutual+gwodegree(0.5))    # Fit the new model
summary(sampmodel.02)

sampmodel.02.gof <- gof(sampmodel.02)                        # Check adequacy
par(mfrow=c(2,2))
plot(sampmodel.02.gof)
par(mfrow=c(1,1))
```

Hmm. The model doesn't fit well at all. What's happening here? Let's look at the degrees directly:

```
degree(samplk3,cmode="outdegree")
```

How is it that literally everyone has 3-4 outgoing ties? This should raise red flags, and take us back to the data source!

In fact, Sampson collected this data by asking respondents to rank their top three alters on each of several dimensions. Thus, in theory everyone should have exactly *three* outgoing ties; however, Sampson didn't entirely adhere to his design, and allowed some subjects to nominate "ties" for their top ranks (identifying four persons overall). It is clear, then, that this data is constrained *by design*, such that each person must have at least three and no more than four outgoing ties. Nothing else could have been observed. Our previous models, however, did not take this constraint into account: they treated all directed graphs on 18 nodes as possible, and thus attempted to capture the degree structure endogenously. To properly model the data, we must constrain the support of our model to match what was actually possible - in this case, the graphs on 18 nodes with outdegrees between three and four.

In **statnet**, support constraints are modeled via the "constraints" argument to `ergm`. Constraints are specified as an implict `formula` object, whose right-hand side specifies the graph properties to be constrained. (By default, all loopless graphs or digraphs with the same number of nodes are possible.) The constraint syntax is flexible, and information on it can be found in `help(ergm)`.

For the present case, we have determined that nodes are constrained to have between three and four outgoing ties. We can specify this by means of the `bd()` (for "bounded degree") term in the constraint formula. `bd` can take many arguments, including `minout` (the minimum outdegree) and `maxout` (the maximum outdegree). Let's fit a model with these constraints:

```
sampmodel.03 <- ergm(samplk3~edges+mutual,constraints=~bd(minout=3,maxout=4))
summary(sampmodel.03)

sampmodel.03.gof <- gof(sampmodel.03)
par(mfrow=c(2,2))
plot(sampmodel.03.gof)
par(mfrow=c(1,1))
```

(Note that the edges term now acts to set the expected proportion of nodes having 3 versus 4 nominations.) The fit is now much better, and more importantly the model is consistent with the data collection process.

Note that we could instead have conditioned on the exact outdegree distribution by using the `outdegreedist` term, e.g.:

```
sampmodel.04 <- ergm(samplk3~mutual,constraints=~outdegreedist)
summary(sampmodel.04)

sampmodel.04.gof <- gof(sampmodel.04)
par(mfrow=c(2,2))
plot(sampmodel.04.gof)
par(mfrow=c(1,1))
```

Here, we are conditioning on the exact number of cases having 3 and 4 outedges (respectively). Although tempting, this is probably not reflective of the data generating process: in theory, since Sampson allowed subjects to nominate four alters rather than three, any or none could have done so. In typical applications, we want our support constraints to reflect what could have been observed.

****

The Sampson data illustrated a relatively simple type of constraint. In other cases, the constraints we employ may be more complex. For instance, consider the Add Health design, in which each respondent was asked to name up to 5 male and female friends (respectively). In this case, each individual may have anywhere from 0 to 5 outgoing ties to male actors, and 0 to 5 ties to female actors; any additional ties are censored. (This design produces many difficult issues for analysis. Please do not collect your data in this way!)

The data file that came with this tutorial has an example from the publicly available version of the Add Health network data, which we will use to illustrate the use of complex support constraints with `ergm`. First, let's load the file (if you haven't already):

```
load("polnet2011.Rdata")                        # Load the file - may need to change directory
```

Among the objects now in memory should be `ah77`:

```
ah77                                            # Show the network object
plot(ah77, vertex.col="grade")                  # Plot it
```

Let's begin by fitting a seemingly plausible unconstrained model:

```
ahmodel.01 <- ergm(ah77~edges+mutual+nodematch("grade",diff=T)+
    nodematch("sex",diff=TRUE)+nodefactor("sex"))
summary(ahmodel.01)
```

An immediate oddity: the MCMC standard errors seem poorly behaved. Let's check the simulations:

```
ahmodel.01.gof <- gof(ahmodel.01)
par(mfrow=c(2,2))
plot(ahmodel.01.gof)
par(mfrow=c(1,1))
```

We note that no one has more than 10 outgoing ties, but the model often simulates otherwise. In this case, we know that both problems are due to the design. Let's fix the model to match our data generating process. We need to tell `ergm` (1) who's male and who's female, and (2) that each node is allowed to send no more than 5 ties to male nodes, and no more than 5 ties to female nodes. To do this, we need to create some special matrices:

```
n <- network.size(ah77)
sexattr <-  cbind(ah77%v%"sex"==1, ah77%v%"sex"==2)  # Create an attribs matrix
sexattr                             # sexattr[i,j]==TRUE if node i has attribute j, else FALSE
maxout <- matrix(5,nr=n,nc=2)       # Create the outdegree constraint matrix
maxout                    # maxout[i,j] is the max ties that i can send to nodes w/attribute j
```

Now, we can fit the above model with constraints, using the `bd` term. We also drop `nodefactor("sex")`, since we know that degree is highly constrained.

```
ahmodel.02 <- ergm(ah77~edges+mutual+nodematch("grade",diff=T)+nodematch("sex",diff=T),
    constraints=~bd(attribs=sexattr,maxout=maxout),interval=200)
summary(ahmodel.02)                                    # Seems better
```

Let's run another check:

```
ahmodel.02.gof <- gof(ahmodel.02)
par(mfrow=c(2,2))
plot(ahmodel.02.gof)
par(mfrow=c(1,1))
```

We've now got the outdegree right - and, moreover, the cross-gender constraint is now being enforced. We are now in a position to begin working on other properties of the model (e.g., the shared partner distribution) by incorporating additional terms, as necessary.

SECTION 6. MODELING WITH MISSING DATA.

Missing data is a significant problem for social network analysis, as most traditional modes of analysis assume that the network of interest is fully observed. In general, the **statnet** packages provide three basic approaches to dealing with missing data:

1) For some types of network descriptives (e.g., density), values are computed based on the observed data, or reasonable fallbacks thereto. This functionality, where available, is usually specified in the help pages of specific functions.

2) For data with multiple indicators per edge, network inference procedures can be used to estimate edge states (even when some information is missing). Several of these can be found in the **sna** package, e.g., the `bbnam` and `consensus` routines.

3) Within **ergm**, some kinds of missingness can be handled via a latent missing data approach (Handcock and Gile, 2007). Essentially a form of tacit multiple imputation, we maximize the likelihood of the observed data under a given model, marginalizing over the unobserved data (with the assumption that this data was produced via the same process).

Here, we focus on (3), although some examples using (2) are also shown. More information on the above can be found in the associated help pages.

Let's begin by considering an artificial case:

```
missnet <- network.initialize(10,directed=F)                # Create an empty network
missnet[1,2] <- missnet[2,7] <- missnet[3,6] <- 1           # Add some edges
missnet[4,6] <- missnet[4,9] <- NA                          # Add some missing edges
missnet                                                     # Missing edges are reported!
missnet[,]                                                  # Also show up in adjacency form
plot(missnet)
```

Note that the **ergm** formula summary command, which gives values for model statistics evaluated on an input graph, tacitly treats missing edges as if they were absent:

```
summary(missnet~edges)                                      # Count edges, leaving out NAs
sum(missnet[,],na.rm=TRUE)/2                                # Same as above
```

ergm, itself, does *not* do this. Instead, it recognizes missing edges, and uses the observed data to model them:

```
ergm(missnet~edges)
```

The coefficient equals -2.590. This is the log odds 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. By contrast, if we treated the missing edges as absent, our coefficient would have been the logit of 3/45, or -2.639.

In some cases, modeling missing edges rather than treating them as absent can have substantial impact on our conclusions. For instance:

```
ergm(missnet~edges+degree(2))                               # Try a model with a degree term

missnet[4,6] <- missnet[4,9] <- 0                           # Now, force the NA cells to 0

ergm(missnet~edges+degree(2))                               # Compare the results
```

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. Such effects can alter our estimates in ways that can be difficult to anticipate without the aid of a model.

****

Although many patterns of missingness are possible, the most common type in social network research is generally the absence of information on outgoing (or occasionally incoming) ties associated with certain nodes, due e.g., to survey non-response. Here, we consider one example of this type, data from David Krackhardt's Silicon Systems study. The study involved 36 persons in a firm undergoing a unionization drive; of the 36, three did not complete the network survey. Although Krackhardt employed a more powerful instrument (a "cognitive social structure" (CSS) instrument, which collected information from all

parties about their own and others' ties), we will focus initially on the "own report" portion of the data. Specifically, we consider the network of self-reported advice-seeking ties, such that *i* has a tie to *j* if *i* indicates that he or she seeks help or advice from *j* at work. The resulting data is as follows:

```
load("polnet2011.Rdata")        # Only needed if you haven't already loaded the workshop data
ssad                            # Silicon Systems advice, "own report" - note the missingness
ssad[,]                         # Missing by rows - three subjects did not respond
ssad%v%"Responded"              # This is also encoded as a vertex attribute
```

For the three non-responding individuals, outgoing ties are unknown; incoming ties, however, are observed for these persons, as are ties among the 33 other subjects. Can we model this data using **ergm**'s latent missing data mechanism? The answer is "yes," so long as the data in question is ignorably missing. Ignorability is a technical concept, that can as a practical matter be understood as the condition under which *the pattern of missingness does not itself convey information of interest, once we control for fully observed covariates*. For purposes of this example, we assume that this condition is satisfied; in general, however, one should be alert for the possibility that missingness is systematic.

As a comparator for the model-based approach, we also compare two "traditional" methods of dealing with non-response: treating NAs as edge-absent, and truncating the data set by eliminating non-responding subjects altogether. We create the corresponding data objects as follows:

```
ssad0 <- ssad                                                   # Force missing edges to absent
ssad0[is.na(ssad[,])] <- 0

ssadtr <- ssad                                                  # Drop non-respondents
delete.vertices(ssadtr, which(!(ssadtr%v%"Responded")))
```

Now, let's fit the same model to each data set. Our model is based primarily on covariates (measures of perceived "charisma," "potency" (i.e., power/effectiveness in the workplace), organizational rank, and membership in the potential bargaining unit for the unionization drive), with the addition of a mutuality effect.

```
ssad.fit<-ergm(ssad~edges+mutual+nodeicov("Charisma")+nodeicov("Rank")+
    nodeicov("Potency")+nodeocov("Charisma")+nodeocov("Rank")+nodeocov("Potency")+
    nodematch("Bargaining.Unit",diff=TRUE))

ssad0.fit<-ergm(ssad0~edges+mutual+nodeicov("Charisma")+nodeicov("Rank")+
    nodeicov("Potency")+nodeocov("Charisma")+nodeocov("Rank")+nodeocov("Potency")+
    nodematch("Bargaining.Unit",diff=TRUE))

ssadtr.fit<-ergm(ssadtr~edges+mutual+nodeicov("Charisma")+nodeicov("Rank")+
    nodeicov("Potency")+nodeocov("Charisma")+nodeocov("Rank")+nodeocov("Potency")+
    nodematch("Bargaining.Unit",diff=TRUE))
```

Note that only the first of these models actually imputes missing edges: the others either treat them as known to be absent, or do not consider any relations involving non-respondents whatsoever. How do the results compare?

```
coefs <- cbind(ssad.fit$coef,ssad0.fit$coef,ssadtr.fit$coef)
colnames(coefs) <- c("Modeled","ForcedZero","Truncated")
round(coefs,3)
```

With a bit more **R** magic, we can also see this graphically:

```
ses<-cbind(diag(ssad.fit$covar),diag(ssad0.fit$covar),diag(ssadtr.fit$covar))^0.5
par(mfrow=c(1,1),mar=c(9,4,2,2))
plot(1,0,type="n",xlim=c(1,NROW(coefs)+0.4),ylim=c(min(coefs-1.96*ses),max(coefs+1.96*ses)),
    axes=FALSE,xlab="",ylab="Coefficient")
abline(h=0,lty=3)
for(i in 1:3){
  points((1:NROW(coefs))+0.2*(i-1),coefs[,i],pch=19,col=i+1,cex=2)
  segments((1:NROW(coefs))+0.2*(i-1),coefs[,i]-1.96*ses[,i],
      (1:NROW(coefs))+0.2*(i-1),coefs[,i]+1.96*ses[,i],lwd=2,col=i+1)
}
axis(2)
axis(1,1:NROW(coefs),labels=rownames(coefs),las=3)
```

```
legend("topleft",legend=c("Modeled","ForcedZero","Truncated"),fill=2:4,bty="n")
```

In this case, most differences are not huge, but they could still be consequential for certain purposes. Moreover, the two deterministic schemes fail to give us any way to assess what might be going on with the non-responding individuals. By contrast, modeling the missing data allows us to get some sense of the unobserved edges via multiple imputation. We can easily accomplish this via simulate, using the `constraints=~observed` setting (which forces the underlying MCMC algorithm to simulate only networks for which the non-missing edge variables match their observed values). Given simulated draws from the complete network, we can network global network properties that could not have been otherwise estimated, and/or properties of the individual positions in question.

```
ssad.sim<-simulate(ssad.fit,constraints=~observed,nsim=100)  # Simulate | observed edges

# Were the observed edges truly preserved?  Trust, but verify!
sapply(ssad.sim$networks,function(z){all(z[!is.na(ssad[,])]==ssad[!is.na(ssad[,])])})

# Some examples of predictive distributions from the model, conditional on the observed
# data; any network descriptive (e.g., those in sna) can be used in this way
par(mfrow=c(2,2))
hist(sapply(ssad.sim$networks,gtrans),main="Predictive Transitivity",xlab="Transitivity")
hist(sapply(ssad.sim$networks,function(z){sum(z[13,])}),
    main="Predictive Degree, Mel",xlab="Degree")
hist(sapply(ssad.sim$networks,function(z){betweenness(z)[24]}),
    main="Predictive Betweenness, Rick",xlab="Betweenness")
hist(sapply(ssad.sim$networks,function(z){evcent(z)[35]}),
    main="Predictive EVCent, Irv",xlab="Eigenvector Centrality")
par(mfrow=c(1,1))


********************************************************************************
```

As noted above, there is a third approach to modeling networks with missing data in **statnet**: use of a network inference (i.e., measurement) model on the raw data, followed by ERGM estimation on the inferred network. While time prohibits extensive coverage of this approach here, we nonetheless provide a brief illustration.

In our previous analysis of the Silicon Systems data, we considered only individuals' reports of their own outgoing ties. As noted, however, Krackhardt collected proxy reports of all ties among all persons from all respondents. Even with three non-respondents, then, we still have a total of 33 reports on each edge that can be used for modeling purposes. This is a great deal of information, that can easily permit highly accurate estimation of the underlying network (even when each individual proxy report is of fairly low quality). Methods such as those implemented in the `consensus` and `bbnam` routines within the **sna** package can integrate such information in a principled way. Here, we employ the Bayesian network inference approach of Butts (2003) to estimate the complete network from the collection of informant reports.

Because the network inference routine takes several minutes to run, we have included a pre-saved set of posterior draws in the workshop data file. However, for those who would like to experiment on their own, the following code will draw from the joint posterior of the network data (using a relatively sparse, diffuse network prior and Beta(1,11) error priors; see `?bbnam`).

```
# No need to run this now, but can do so later!
#ssad.post<-bbnam(silsys.ad.css,nprior=0.2,draws=300,burntime=100,reps=3)   # 300 draws
```

Each posterior draw is associated with a complete network; by fitting an ERGM to each draw, we can examine the posterior predictive distribution of the ERGM coefficients. This distribution takes into account our uncertainty regarding the underlying network, stemming both from missing data and from measurement error (i.e., incorrect responses). Because this process is time consuming, it too has been pre-computed for purposes of this exercise. The code used to do so is as follows.

```
# No need to run this now, but can do so later!
#basenet<-silsys.ad.css[[1]]    # For convenience, create an "empty" base network w/covariates
#basenet[,]<-0
#
#ssad.fit.pp<-npostpred(ssad.post,function(z,basenet){  # Draw from the posterior predictive
#  net<-basenet;
#  net[,]<-z;
#  ergm(net~edges+mutual+nodeicov("Charisma")+nodeicov("Rank")+
#      nodeicov("Potency")+nodeocov("Charisma")+nodeocov("Rank")+nodeocov("Potency")+
```

```
#       nodematch("Bargaining.Unit",diff=TRUE))$coef
#},basenet=basenet)
```

Although it is usually wise to bring forward the uncertainty of the full posterior distribution where feasible, another alternative is to select a posterior estimate of the network, and fit an ERGM to that. One very simple estimator that can be employed is the edgewise marginal mode, itself equivalent to the central graph of the posterior draws; this is the network in which the *i,j* edge is taken to be present if the marginal posterior probability of an *i,j* edge is greater than 0.5. This is easily computed from the set of posterior draws, and we employ it below.

```
basenet<-silsys.ad.css[[1]]                 # If you didn't run this earlier, run it now
basenet[,]<-0

ssad.pmm <- basenet   # Begin with our empty "base graph"
ssad.pmm[,] <- centralgraph(ssad.post$net)  # Add edges based on the central graph of the
                                            # posterior draws
# Now, model the result in the usual way
ssad.fit.pmm <- ergm(ssad.pmm~edges+mutual+nodeicov("Charisma")+nodeicov("Rank")+
    nodeicov("Potency")+nodeocov("Charisma")+nodeocov("Rank")+
    nodeocov("Potency")+nodematch("Bargaining.Unit",diff=TRUE))
```

How do these methods compare? Let's consider the posterior mean coefficients, the coefficients based on the edgewise marginal mode, and the coefficients we obtained from our three previous techniques.

```
coefs<-cbind(rowMeans(ssad.fit.pp),ssad.fit.pmm$coef,ssad.fit$coef,ssad0.fit$coef,
    ssadtr.fit$coef)
colnames(coefs)<-c("MeasPMean","MeasPMarMode","Modeled","ForcedZero","Truncated")
round(coefs,3)
```

As before, we can also visualize:

```
par(mar=c(9,4,2,2))
boxplot(t(ssad.fit.pp),las=3)
abline(h=0,lty=3)
for(i in 2:5)
  points(1:NROW(coefs),coefs[,i],col=i,pch=19,cex=2)
legend("bottomright",fill=2:5,legend=colnames(coefs)[-1],bty="n")
```

We can see that the posterior marginal mode estimate is quite close to the posterior mean of the coefficients, and the distributions are not especially wide: the measurement model in this case has a great deal of data to employ, and has little uncertainty regarding the network structure. On the other hand, the two measurement-based estimators are substantially different from the estimators based on the own-report data (including the model-based imputation method). Why the difference? The key lies in the amount and quality of information available. Single informant reports are not always accurate (even regarding one's own ties!), and the models based on own-report must take these reports at face value. In the case of missing data, the situation is even worse: now, the model must impute the missing ties from covariate and dependence effects, which are at best imperfectly informative. The net result is a noisier, less informative basis from which to infer ERGM coefficients.

Of course, to use these network inference methods requires that we obtain more than one observation per edge - something that is not done in most current network studies. When collecting new data from archival materials, human informants, or other information sources prone to error and missingness, we strongly encourage designs that collect multiple observations for each potential relationship. Although such data can be more costly to collect, the potential quality gain (and robustness to missing data) can be substantial.

SECTION 7. ADDITIONAL FUNCTIONALITY

7.1. Additional functionality

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

- analysis of bipartite networks (**network** 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)
- simulation of bipartite networks with given degree distributions (**networksis** package)
- tools for fitting relational event models (**relevent** package)
- hierarchical ERGMs (**hergm** package)

Any of these additional packages can be downloaded from CRAN. For more detailed information on these features, please visit the **statnet** webpage (http://statnet.org).

7.2. Additional functionality in development:

- dynamic network estimation and simulation (**ergm** package) – expected later 2011
- hierarchical versions of relational event models (**relevent** package) – expected later 2011
- ERGMs for valued ties (**ergm** package) – expected later 2011
- model-based simulation of complete networks from egocentric network data – expected 2012

7.3. Statnet Commons: The development group

APPENDIX A: The many references of 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-family Random Graph Model; any statistical model for relational data that is written in exponential family form.
- **ergm package**: one of the constituent 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 constituent 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

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		