`statnet` Tutorial

Carter Butts (University of California, Irvine)
Martina Morris (University of Washington)
Nicole Carnegie (University of Washington)
Pavel Krivitsky (University of Washington)

Joint with the rest of the Statnet Development Team:
Mark S. Handcock (University of Washington)
David R. Hunter (Penn State University)
Steve M. Goodreau (University of Washington)

INSNA Sunbelt workshop February 2009

Table of contents

Basic resources

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.

VERSIONS OF SOFTWARE USED IN THIS TUTORIAL:

R: 2.8.1
statnet: 2.2.1

SECTION 0. GETTING STARTED

*0.1. Download and install the latest version of **R**. (R 2.8.1 at the time of the Sunbelt 2009 workshop)*

a. Go to http://cran.r-project.org/, and select Mirrors from the left-hand menu.
b. Select a location near you.
c. From the "Download and Install **R**" section, select the link for your operating system.
d. Follow the instructions on the relevant page.
e. Note that you need to download only the base distribution, not the contributed packages.
f. Once you've downloaded the installation file, follow the instructions for installation.

0.2. Download and attaching **statnet** *and associated packages:*

a. Open **R**.
b. Install the **statnet** installer. At the **R** cursor >, type:

```
install.packages("statnet")
```

c. Now, and in the future, you can install/update **statnet** at any point using the installer that comes with **statnet**. Step (b) is only necessary the first time you wish to use **statnet** but does not need to be repeated each time. At the **R** cursor, type:

```
update.statnet()
```

Follow the directions; feel free to say no to any optional packages, although we recommend saying yes.
The first choice provided is to install all the required and optional packages.

d. Attach **statnet** to your **R** session by typing:

```
library(statnet)
```

0.3. Set a specific working directory for this tutorial if you wish.

a. If you are using Windows, go to File > Change Dir and choose the directory.
b. Otherwise, use the setwd directory command from the **R** cursor >:

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

0.4. Install another package we will be using in this tutorial.

```
install.packages('coda')                # install package from CRAN
library(coda)                           # attach installed package
```

SECTION 1. WORLD'S SHORTEST R TUTORIAL

*1.1. A few facts to remember about **R**.*

- **R** mostly runs through the command line or through batch files. However, one can perform basic file management through the menu in the Windows version.
- Everything in **R** is an object, including data, output, functions—everything.
- Objects that are created during a session are saved in the "global environment" by default, which is stored as a single file (".RData") in the working directory.
- **R** is case sensitive.
- **R** comes with a set of pre-loaded functions. Others can be added by downloading from the **R** project website as we did above. Downloaded packages should be updated periodically. To use the package in your session, it must be attached using the *library* command.

1.2. An extremely brief R tutorial

```
a <- 3                                  # assignment
a                                       # evaluation
[1] 3

sqrt(a)                                 # perform an operation
b <- sqrt(a)                            # perform operation and save
b

ls()                                    # list objects in global environment
help(sqrt)                              # help w/ functions
?sqrt                                   # same thing
help.search('square')                   # hip to the square (root)?
??square                                # same thing
help.start()                            # lots of help

rm(a)                                   # remove an object

# Use the File menu to save your current global environment, change working
# directory, exit, etc.
```

SECTION 2: A QUICK INTRODUCTION TO `network`: DATA IMPORT, EXPLORATION, MANIPULATION

2.1 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
```

2.2 Importing Relational Data

```
# First find the files "flo.paj" and "floadj.txt" that come with the package
flo.paj.location <- paste(.find.package("statnet"),"flo.paj",sep="/")
flo.paj.location
floadj.txt.location <- paste(.find.package("statnet"),"floadj.txt",sep="/")
floadj.txt.location

#Read in an adjacency matrix
floadj <- read.table(floadj.txt.location,header=TRUE)
floadj

#Read in a Pajek file
flopaj <- read.paj(flo.paj.location)
class(flopaj)                                 #What class of object is this?  It's a list
names(flopaj)                       #This is a project file, with networks and other data
names(flopaj$networks)                              #See which networks are in the file
nflo2 <- flopaj$networks[[1]]                                #Extract the marriage data
class(nflo2)                                              #What class of object is this?
nflo2                                                        #Examine the network object
```

2.3 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                                                             #Compare with nflo
```

2.4 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
```

2.5 Sample **sna** *Routines*

```
library(sna)                                                          #Load the sna library
library(help='sna')                                                #See also this for a list

betweenness(flo)
isolates(nflo)
gplot(nflo,displaylabels=T)
gplot3d(nflo,displaylabels=T)
kcycle.census(nflo,mode='graph',maxlen=5)
```

2.6 Advanced Commands (Peruse at your leisure)

Working With Edges

```
#Adding edges
g <- network.initialize(5)                          #Create an empty graph
g[1,2] <- 1                                       #Add an edge from 1 to 2
g[2,] <- 1                                  #Add edges from 2 to everyone else
g                                                     #Examine the result


#Delete edges
g[1,2] <- 0                                  #Remove the edge from 3 to 5
g                                                             #It's gone!
g[,] <- 0                                        #Remove all edges
g                                               #Now, an empty graph

#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

#Setting edge values
m <- matrix(1:16^2, nrow=16, ncol=16)        #Create a matrix of edge 'values'
nflo %e% 'boo' <- m                                 #Value the marriage ties

#Retrieving edge values
list.edge.attributes(nflo)                             #See what's available
nflo %e% 'boo'                                         #Use the %e% operator

#For more information....
?network.extraction
```

Working With Vertex Attributes

```
#Add vertex attributes
nflo %v% 'woo' <- letters[1:16]                         #Letter the families
nflo
list.vertex.attributes(nflo)                         #List all vertex attributes
nflo %v% 'woo'                                     #Retrieve the vertex attribute
```

SECTION 3. 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

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

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

Iteration 1 of at most 3: the log-likelihood improved by 0.001783
Iteration 2 of at most 3: the log-likelihood improved by 0.0007659
Iteration 3 of at most 3: the log-likelihood improved by 0.0005095

summary(flomodel.02)

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

Formula:   flomarriage ~ edges + triangle

Newton-Raphson iterations:  4
MCMC sample of size 10000

Monte Carlo MLE Results:
         Estimate Std. Error MCMC s.e. p-value
edges     -1.6990     0.2017     0.050  <1e-04 ***
triangle   0.2043     0.2643     0.032   0.441
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

    Null  Deviance: 166.355  on 120  degrees of freedom
 Residual Deviance: 107.872  on 118  degrees of freedom
          Deviance:  58.484  on   2  degrees of freedom

AIC: 111.87    BIC: 117.45
```

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.miss"    "iterations"
 [5] "MCMCtheta"      "loglikelihood"  "gradient"       "covar"
 [9] "samplesize"     "failure"        "mc.se"          "newnetwork"
[13] "burnin"         "interval"       "network"        "theta.original"
```

```
[17] "mplefit"        "parallel"       "null.deviance"  "mle.lik"
[21] "etamap"         "formula"        "constraints"    "prop.weights"
[25] "offset"         "drop"

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

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 degree centrality is 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 tendency for ties to be
summary(sampmodel.01)                       # reciprocated ("mutuality")


data(faux.mesa.high)                        # Let's try a larger network
mesa <- faux.mesa.high
plot(mesa)
summary(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")                # Table of ties by race of each vertex
```

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.

SECTION 4. NETWORK STATISTICS AVAILABLE FOR *ERGM* MODELING and SIMULATION

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

```
help('ergm-terms')
help('ergm-terms', chmhelp=FALSE)
```

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)

These ergm-term names are used in:

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)

A handy table of terms

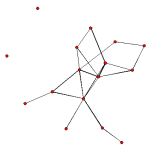
Term	Undir?	Dir?	Bip?	Required Args	Optional Args
Basic terms					
edges	X	X	X		
density	X	X	X		
mutual		X			
asymmetric		X			
meandeg	X	X	X		
Nodal attribute terms					
nodecov	X	X	X	attrname	
nodefactor	X	X	X	attrname	
nodeifactor		X		attrname	
nodeofactor		X		attrname	
nodeicov		X		attrname	
nodeocov		X		attrname	
nodemix	X	X	X	attrname	
nodematch	X	X	X	attrname	
absdiff	X	X	X	attrname	
smalldiff	X	X	X	attrname, cutoff	
b1factor			X	attrname	base
b2factor			X	attrname	base
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
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
m2star		X			
altkstar	X	X	X	lambda	fixed
Cycle and triangle terms					
triangle	X	X			attrname
ctriple		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			
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					
twopath	X	X	X		
threepath	X	X	X		

SECTION 5. 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 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.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

 Nodal attribute names:
    vertex.names wealth priorates totalties

 edgelist matrix:
      [,1] [,2]
 [1,]    3    1
 [2,]    9    1
  .... [output cut].

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

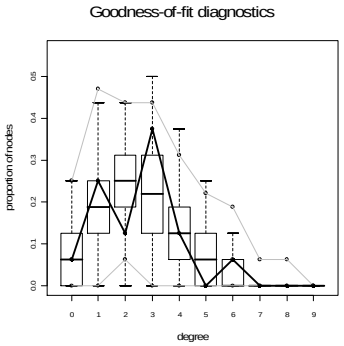


Voilà. (Of course, yours will look somewhat different.)

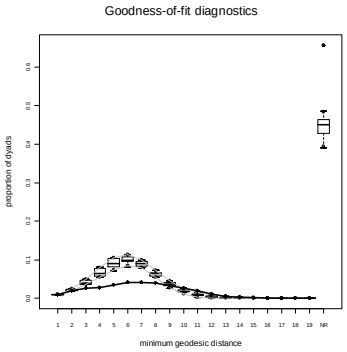
SECTION 6. EXAMINING THE QUALITY OF MODEL FIT – *GOF*.

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)
```




```
fauxmodel.02 <- ergm(mesa~edges)
fauxmodel.02.gof <- gof(fauxmodel.02~distance,nsim=10)
plot(fauxmodel.02.gof)
```



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

SECTION 7. 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 www.statnet.org)

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), interval=1, burnin=1000, seed=2)
```

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=.75)

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

You can get more feedback about this during the fitting, with the "verbose" argument:

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

You might try to increase the MCMC sample size:

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

The line in the feedback about the network having more than 50,000 edges tells us we have headed of towards a full graph for sure. So a larger MCMC sample is not the answer.

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),seed=1)
mcmc.diagnostics(fit)
```

Still degenerate, but maybe getting closer? Try adding some nodal covariates.

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

Send the plots to a file rather than the screen, since there will be a few pages now.

```
pdf('diagnostics.pdf')
mcmc.diagnostics(fit)
dev.off()
```

Look at the plots by opening the file. Looks a bit better; try reducing the fixed gwesp parameter:

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

pdf('diagnostics.pdf')
mcmc.diagnostics(fit)
dev.off()
```

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

SECTION 8. ADDITIONAL FUNCTIONALITY

8.1. Additional functionality

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

- latent space and latent cluster analysis (the **latentnet** package)
    - *see Pavel Krivitsky's talk Thursday 4:30*
- analysis of bipartite networks (**network** package)
- network permutation models (**netperm** package)
- MLE estimation for degree distributions (negative binomial, Poisson, scale-free, etc.) (**degreenet** package)
- curved exponential family estimation and simulation (**ergm** base package)
- Simulation of bipartite networks with given degree distributions (**networksis** 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).

8.2. Adding new change statistics

Statnet allows users to code additional network statistics that are not included in the downloaded software. For further instructions, visit: http://statnet.org

8.3. Additional functionality in development:

- dynamic network estimation and simulation (**statnet** base package)
    - *see Pavel Krivitsky's talk Friday 8:30am*
- analysis of network samples and networks with missing data (**statnet** base package)
- simulation and analysis of complete networks from egocentric network data (**egonet** package)

8.4. 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 Random Graph Model; a statistical model for relational data that takes a generalized exponential 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.