Moving Beyond Descriptives: An Introduction to Basic Network Statistics with statnet

2011 Sunbelt Social Networks Conference
St. Pete's Beach, Florida
February 2011

Presenters:
Carter T. Butts (University of California, Irvine)
Ryan M. Acton (University of Massachusetts Amherst)
Lorien Jasny (University of California, Irvine)
Zack Almquist (University of California, Irvine)

statnet Core Development Team:
Carter T. Butts (University of California, Irvine)
Steven M. Goodreau (University of Washington)
Mark S. Handcock (University of Washington)
David R. Hunter (Penn State University)
Martina Morris (University of Washington)

Table of contents **Author**

Basic resources

R webpage: http://www.r-project.org
Helpful **R** tutorials: http://cran.r-project.org/other-docs.html
statnet webpage: http://statnet.org
statnet help: statnet_help@statnet.org
Workshop web site: http://statnet2011.statnet.org/

Typographical conventions

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

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

All other text represents instructions and guidance.

SECTION 0. GETTING STARTED

**** The instructions in Section 0 match those on the workshop web page (http://sunbelt2011.statnet.org/). If you have already installed the required software, there is no need to do so again. ****

0.1. Download and install the latest version of **`R`**

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.packages("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. Download and install supplemental packages:

a. We recommend installing some additional, non-**`statnet`** packages that are employed in selected exercises; you do not have to install these packages to use **`statnet`** in general, but some specific functions (or other analyses shown here) do expect them. To do so, at the **`R`** cursor type:

```
install.packages(c("numDeriv","yacca","rgl"))
```

Note: **`rgl`** sometimes has problems installing on specific platforms. If you can't install it, don't worry -- you need it for 3D network visualization, but not for anything else. Consult the **`R`** web site for more information on these or other contributed packages.

0.4. 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:

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

SECTION 1. VISUALIZATION AND DESCRIPTIVES

1.1 Getting started

```
library(sna)                      # Load the sna library
library(help="sna")               # See also this for a list
load("sunbelt2011.Rdata")         # Load supplemental workshop data

#For more information....
?help.start
?library
?sna
```

1.2 Network visualization with gplot

```
# Begin by plotting contiguity among nations in 1993 (from the Correlates of War project)
gplot(contig_1993, usearrows=FALSE)            # Visualizing, but turn off arrows manually
gplot(contig_1993, gmode="graph")              # Can also tell gplot the data is undirected

# Here's an example of directed data - militarized interstate disputes (MIDs) for 1993
gplot(mids_1993, label.cex=0.5, label.col=4,
   label=network.vertex.names(mids_1993))      # Basic display, with labels

# All those isolates can get in the way - we can suppress them using displayisolates
gplot(mids_1993, displayisolates=FALSE, label.cex=0.5, label.col=4,
   label=network.vertex.names(mids_1993))

# The default layout algorithm is that of Frutchterman-Reingold (1991), can use others
gplot(mids_1993, displayisolates=FALSE, label.cex=0.5, label.col=4,
   label=network.vertex.names(mids_1993), mode="mds")     # MDS of position similarity

# When a layout is generated, the results can be saved for later reuse:
coords <- gplot(contig_1993)                    # Capture the magic of the moment
coords                                          # Show the vertex coordinates

#Saved (or a priori) layouts can be used via the coord argument:
gplot(mids_1993, label.cex=0.5, label.col=4, coord=coords,
   label=network.vertex.names(mids_1993))       # Relive the magic

# When the default settings are insufficient, interactive mode allows for tweaking
coords <- gplot(contig_1993, interactive=TRUE)   # Modify and save
gplot(contig_1993, coord=coords, gmode="graph")  # Should reproduce the modified layout

#For more information....
?gplot
?gplot.layout
```

1.3 Three-dimensional visualization with gplot3d (requires the rgl package)

```
gplot3d(contig_1993, label=network.vertex.names(contig_1993))   # Experience the future!
# Other layouts are possible here, too:
gplot3d(contig_1993, label=network.vertex.names(contig_1993), mode="kamadakawai")

#For more information....
?gplot3d
?gplot3d.layout
```

1.4 Basic centrality indices (degree, betweenness, and closeness)

```
# We begin with the simplest case: degree
degree(mids_1993)                                              # Default: total degree
```

```
ideg <- degree(mids_1993, cmode="indegree")                  # Indegree for MIDs
odeg <- degree(mids_1993, cmode="outdegree")                 # Outdegree for MIDs

# Once centrality scores are computed, we can handle them using standard R methods:
plot(ideg, odeg, type="n", xlab="Incoming MIDs", ylab="Outgoing MIDs") # Plot ideg by odeg
abline(0, 1, lty=3)
text(jitter(ideg), jitter(odeg), network.vertex.names(contig_1993), cex=0.75, col=2)

#Plot simple histograms of the degree distribution:
par(mfrow=c(2,2))                                           # Set up a 2x2 display
hist(ideg, xlab="Indegree", main="Indegree Distribution", prob=TRUE)
hist(odeg, xlab="Outdegree", main="Outdegree Distribution", prob=TRUE)
hist(ideg+odeg, xlab="Total Degree", main="Total Degree Distribution", prob=TRUE)
par(mfrow=c(1,1))                                           # Restore display

# Centrality scores can also be used with other sna routines, e.g., gplot
gplot(mids_1993, vertex.cex=(ideg+odeg)^0.5/2, vertex.sides=50,
    label.cex=0.4, vertex.col=rgb(odeg/max(odeg),0,ideg/max(ideg)),
    label=network.vertex.names(mids_1993))

# Betweenness and closeness are also popular measures
bet <- betweenness(contig_1993, gmode="graph")             # Geographic betweenness
bet
gplot(contig_1993, vertex.cex=sqrt(bet)/25, gmode="graph")   # Use w/gplot
clo <- closeness(contig_1993)                              # Geographic closeness
clo                                                        # A large world after all?

#For more information....
?betweenness
?bonpow
?closeness
?degree
?evcent
?graphcent
?infocent
?prestige
?stresscent
```

1.5 From centrality to centralization

```
centralization(mids_1993, degree, cmode="indegree")         # Do MIDs concentrate?
centralization(contig_1993, evcent)                         # Eigenvector centralization

#For more information....
?centralization
```

SECTION 2. NODE LEVEL INDICES, NETWORK COVARIATES, AND NETWORK REGRESSION

2.1 Getting Started

```
library(network)                                   #Load network if needed
data(emon)                                         #Load Drabek et al. data

#Extract ties from the Cheyenne EMON communicating at least "every few hours"
g<-as.sociomatrix(emon[[1]],"Frequency")           #Need to get the frequency info
g<-symmetrize((g>0)&(g<4))                         #Note the reverse coding!
```

2.2 Initial Analysis

```
#Get some potential covariates
drs<-emon[[1]]%v%"Decision.Rank.Score"             #Get decision rank (see man page)
crs<-emon[[1]]%v%"Command.Rank.Score"              #Get command rank

#Calculate some basic centrality measures
deg<-degree(g,gmode="graph")
bet<-betweenness(g,gmode="graph")
clo<-closeness(g,gmode="graph")

#Raw correlations
cor(cbind(deg,bet,clo),cbind(drs,crs))

#Classical tests (using asymptotic t distribution)
cor.test(deg,drs)
cor.test(bet,drs)
cor.test(clo,drs)
```

2.3 Testing correlations

```
#Permutation tests
perm.cor.test<-function(x,y,niter=5000){  #Define a simple test function
  c.obs<-cor(x,y,use="complete.obs")
  c.rep<-vector()
  for(i in 1:niter)
    c.rep[i]<-cor(x,sample(y),use="complete.obs")
  cat("Vector Permutation Test:\n\tObserved correlation: ",c.obs,"\tReplicate quantiles
(niter=",niter,")\n",sep="")
  cat("\t\tPr(rho>=obs):",mean(c.rep>=c.obs),"\n")
  cat("\t\tPr(rho<=obs):",mean(c.rep<=c.obs),"\n")
  cat("\t\tPr(|rho|>=|obs|):",mean(abs(c.rep)>=abs(c.obs)),"\n")
  invisible(list(obs=c.obs,rep=c.rep))
}
perm.cor.test(deg,drs)                        #Non-parametric tests of correlation
perm.cor.test(bet,drs)
perm.cor.test(clo,drs)

#For more information....
?emon
?cor.test
?t.test
?sample
```

2.4 Using NLIs as regression covariates

```
pstaff<-emon[[1]]%v%"Paid.Staff"                    # Get more EMON covariates
vstaff<-emon[[1]]%v%"Volunteer.Staff"
govt<-((emon[[1]]%v%"Sponsorship")!="Private")

#Very simple model: decision rank is linear in size, degree, and govt status
mod<-lm(drs~deg+pstaff+vstaff+govt)
summary(mod)
anova(mod)                                           #Some useful lm tools
AIC(mod)

#Try with alternative measures....
mod2<-lm(drs~bet+pstaff+vstaff+govt)                  #Betweenness
summary(mod2)
mod3<-lm(drs~clo+pstaff+vstaff+govt)                  #Closeness
summary(mod3)
AIC(mod,mod2,mod3)                                    #Closeness wins!

#For more information....
?lm
?anova
?AIC
```

2.5 Graph correlation and bivariate QAP

```
# Remember the Florentine families data
data(florentine)
plot(flobusiness)                                # Examine business ties
plot(flomarriage)                                # Examine marriage ties

# Could the two be related?  Let's try a graph correlation
gcor(flobusiness,flomarriage)

# To test the correlation, we can use the qaptest routine
qt<-qaptest(list(flobusiness,flomarriage),gcor,g1=1,g2=2)
summary(qt)                                      # Examine the results
plot(qt)                                         # Plot the QAP distribution

# Testing against covariate effects
wealth<-sapply(flomarriage%v%"wealth",rep,network.size(flomarriage))
wealthdiff<-abs(outer(flomarriage%v%"wealth",flomarriage%v%"wealth","-"))
qt1<-qaptest(list(flomarriage,wealth),gcor,g1=1,g2=2)
qt2<-qaptest(list(flomarriage,wealthdiff),gcor,g1=1,g2=2)
summary(qt1)                                     # Do wealthy families have more ties?
summary(qt2)                                     # Is there a wealth difference effect?

# For more information....
?qaptest
?gcor
?outer
?sapply
?rep
```

2.6 Network regression

```
# We begin by preparing the response variable.  We will use the Cheyenne
# EMON in valued form, but need to recode the frequency data
data(emon)
Y<-as.sociomatrix(emon[[1]], "Frequency")        # Extract frequencies
Y[Y>0]<-5-Y[Y>0]                                 # Now, higher -> more frequent

# Extract some vertex attributes
crk<-emon[[1]]%v% "Command.Rank.Score"              # Command rank
spon<-emon[[1]]%v%"Sponsorship"                     # Type of organization
```

```
# Form some predictor matrices (X variables)
Xcr<-sapply(crk,rep,length(crk))                          # Column effects for command rank
Xcr
Xsp<-outer(spon,spon,"!=")                                # Dyadic effect for type difference
Xsp

# Fit the model (takes a while to perform QAP test)
cmfit<-netlm(Y,list(Xcr,Xsp))
summary(cmfit)                                            # Examine the results

# For more information....
?outer
?netlm
```

SECTION 3. GRAPH LEVEL INDICES AND CONDITIONAL UNIFORM GRAPH TESTS

3.1 Permutation tests for GLI/graph-level covariate association

```
# Here we consider the famous Sampson monastery data:
par(mfrow=c(4,3),mar=c(2,1,4,1))
for(i in 1:length(sampson))
  gplot(sampson[[i]],displaylabel=TRUE,boxed.label=FALSE,main=names(sampson)[i])

# Are positive relations more reciprocal (relative to density) than negative
# ones?  Let's try a simple permutation test:
r4<-grecip(sampson,measure="edgewise.lrr")
ispos<-c(TRUE,FALSE,TRUE,FALSE,TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)
obs<-sum(r4[ispos])-sum(r4[!ispos])
reps<-vector()
for(i in 1:1e4){
  temp<-sample(ispos)
  reps[i]<-sum(r4[temp])-sum(r4[!temp])
}
mean(reps>=obs)                                           # Upper tail p-value
mean(abs(reps)>=abs(obs))                                 # Two-sided version
hist(reps)
abline(v=obs,col=2,lwd=3)                                 # Visualize it

# We can look at transitivity as well.  How does transitivity compare to density?  (Log-odds
# method)
log(gtrans(sampson)/gden(sampson))

# Are positive relations more transitive (relative to density) than negative
# ones?  Let's try another vector permutation test:
ltr<-log(gtrans(sampson)/gden(sampson))
obs<-sum(ltr[ispos])-sum(ltr[!ispos])
reps<-vector()
for(i in 1:1e4){
  temp<-sample(ispos)
  reps[i]<-sum(ltr[temp])-sum(ltr[!temp])
}
mean(reps>=obs)                                           # Upper tail p-value
mean(abs(reps)>=abs(obs))                                 # Two-sided version
hist(reps)
abline(v=obs,col=2,lwd=3)                                 # Visualize it
```

3.2 Comparing graphs via the triad census

```
# Let's get the triad census for each network
tc<-triad.census(sampson)
tc

# Cool trick: two-way correspondence analysis of graphs and their triad census
# scores (aka a "Faust diagram").  Networks here appear close to the triad
# types they contain at excess frequency (distances are chi-squared based;
# see references in ?corresp for more detail).
library(MASS)                                    # Requires the MASS package
plot(corresp(tc,nf=2))                           # Plot network/triad association

# What if this data were symmetric?  We can symmetrize to illustrate.
tc<-triad.census(symmetrize(sampson),mode="graph")#Need to use mode="graph" here
rownames(tc)<-names(sampson)
plot(corresp(tc,nf=2))                           # Plot undirected network/triad association

# For more information....
?gtrans
?triad.census
?corresp
?symmetrize
```

3.3 Simple univariate conditional uniform graph tests

```
# The cug.test function provides a simple interface for univariate CUG tests.
# Let's try testing some data on trade in complex manufactured goods to see if overall
# activity (density) is greater then would be expected from size alone.
cug.test(ctrade,gden)                    # Call cug.test with the gden (density) function

# Is there more reciprocity than density would suggest?  Let's see.
cug.test(ctrade,grecip,cmode="edges")    # Conditioning on edges, calling grecip

# Given biases in density and reciprocity, we might look to see if there is a
# bias in transitivity, too.  This time, let's condition on all of the above.
cug.test(ctrade,gtrans,cmode="dyad")     # Conditioning on dyad census

# We are not limited to simple commands.  Let's try indegree centralization:
ct<-cug.test(ctrade,centralization,cmode="dyad",FUN.arg=list(FUN=degree,
    cmode="indegree"))  # Note that we here must pass not only arguments to
                        # cug.test, but also to centralization and degree!
ct                                                          # Print the result
plot(ct)                                                    # Can also plot it!
```

SECTION 4. MULTIVARIATE ANALYSIS OF GRAPH SETS

4.1 Distance based methods: clustering and scaling

```
# Start by calculating Hamming distances for the Sampson data
samphd<-hdist(sampson)
samphd

# Now, try an MDS solution
sampmds<-cmdscale(samphd)
sampmds
plot(sampmds,type="n")                                  # Plot the results
text(sampmds,label=names(sampson))

# MDS suggests a three-cluster solution; let's try hclust
samphc<-hclust(as.dist(samphd))
plot(samphc,labels=names(sampson))                      # Very clear solution
rect.hclust(samphc,k=3)

# Examine central graphs for each cluster
sampcg<-gclust.centralgraph(samphc,k=3,sampson)
par(mfrow=c(2,2))
gplot(sampcg[1,,],main="Positive CG")
gplot(sampcg[2,,],main="Negative CG")
gplot(sampcg[3,,],main="Liking CG")
par(mfrow=c(1,1))

# More fun - can plot stats by cluster!
gclust.boxstats(samphc,k=3,grecip(sampson,measure="edgewise.lrr"), names=c("Positive",
    "Negative","Liking"), main="Edgewise LRR Reciprocity by Relational Type")
gclust.boxstats(samphc,k=3,gtrans(sampson), names=c("Positive","Negative","Liking"),
    main="Transitivity by Relational Type")

# Can also plot stats by MDS
gdist.plotstats(samphd,cbind(grecip(sampson,measure="edgewise.lrr"),
    gtrans(sampson),centralization(sampson,degree,cmode="indegree")))
legend("bottom",legend=c("Reciprocity","Transitivity","Indegree Cent."),col=1:3,lty=1)

# For more information....
?hdist
?cmdscale
?hclust
?rect.hclust
?gclust.centralgraph
?gdist.plotstats
```

4.2 Network PCA

```
# To begin, let's get the graph correlation matrix for the Sampson nets

sampcor<-gcor(sampson)
sampcor

# Now, we compute the eigendecomposition (could also have used gcov above)
sampeig<-eigen(sampcor)

# Eigenvalues contain variance explained, to whit:
evals<-sampeig$value                                        # Extract eigenvalues
evals/sum(evals)                                            # Variance explained
# Show this as a scree plot
barplot(evals/sum(evals),names.arg=1:length(evals))

# Examine loadings (eigenvectors); looks like first 3 are key
load<-sampeig$vector[,1:3]
rownames(load)<-names(sampson)
load
```

```
# Can rotate using varimax
varimax(load)

# Try plotting the first two components
plot(load[,1:2],type="n",asp=1,xlab="PC 1",ylab="PC 2")
abline(h=0,v=0,lty=3)
arrows(0,0,load[,1],load[,2],col=2)          # Should be read angularly!
text(load[,1:2],label=names(sampson))

# Finally, extract scores
S1<-apply(sweep(as.sociomatrix.sna(sampson),1,load[,1],"*"),c(2,3),sum)
S2<-apply(sweep(as.sociomatrix.sna(sampson),1,load[,2],"*"),c(2,3),sum)
S3<-apply(sweep(as.sociomatrix.sna(sampson),1,load[,3],"*"),c(2,3),sum)

#Visualize a score graph (not too helpful in this case!)
coord<-gplot.layout.fruchtermanreingold(as.edgelist.sna(S1>0),NULL)
gplot(S1!=0,edge.col=sign(S1)+3,coord=coord)

# For more information....
?gcor
?gcov
?eigen
?varimax
?abline
?arrows
?sweep
?gplot.layout
```

4.3 Network CCA

```
# For this one, we're going to use the country trade data.  Somewhat perversely,
# we're going to use yacca instead of netcancor (b/c this data set has
# missing data which netcancor doesn't handle, and b/c yacca currently has
# better visualizations).

library(yacca)                          # You'll need to install this....

# Perform a canonical correlation analysis between relations and attribute
# differences
trade.cca<-cca(gvectorize(trade),gvectorize(tradediff),
    standardize.scores=FALSE)           # Turn off standardization b/c of NAs
summary(trade.cca)

# Visualize the output
plot(trade.cca)

# For more information
?netcancor
?yacca
?cca
?gvectorize
```

4.4 Studying Qualitative Dynamics with Network MDS

```
# Start by calculating Hamming distances for each of the two Johnson data sets

jp1d <- hdist(johnsonPolarY1)
jp2d <- hdist(johnsonPolarY2)

# Now try an MDS solution

jp1mds<-cmdscale(jp1d)
jp1mds
plot(jp1mds,type="l",lty=2)                               # Plot the results
text(jp1mds,label=rownames(johnsonPolarY1),font=2)
```

```
jp2mds<-cmdscale(jp2d)
jp2mds
plot(jp2mds,type="l",lty=2)                             # Plot the results
text(jp2mds,label=rownames(johnsonPolarY2),font=2)
```

APPENDIX 1. A BRIEF 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. Downloaded packages to be used must be attached using the *library* command during any **R** session in which they are to be used.

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

1.2. Introduction to basic **R** *syntax*

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

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

a == a                                         # A is A?
a != b                                         # A is not B

ls()                                          # list objects in global environment
help(sqrt)                                    # help w/ functions
?sqrt                                         # same thing
help.start()                                  # lots of help
help.search("sqrt")                           # what am I looking for?
apropos("sqr")                                # it's on the tip of my tongue...

rm(a)                                         # remove an object
```

1.3. Vectors and matrices in **R**

```
# Creating vectors using the concatenation operator
a <- c(1,3,5)                                 # create a vector by concatenation
a
a[2]                                          # select the second element
b <- c("one","three","five")                  # also works with strings
b
b[2]
a <- c(a,a)                                   # can apply recursively
a
a <- c(a,b)                                   # mixing types - who will win?
a                                             # there can be only one!

# Sequences and replication
a <- seq(from=1,to=5,by=1)                    # from 1 to 5 the slow way
b <- 1:5                                      # a shortcut!
a==b                                          # all TRUE
rep(1,5)                                      # a lot of 1s
rep(1:5,2)                                    # repeat an entire sequence
rep(1:5,each=2)                              # same, but element-wise
rep(1:5,times=5:1)                           # can vary the count of each element

# Any and all (with vectors)
a <- 1:5                                     # create a vector
a>2                                          # some TRUE, some FALSE
any(a>2)                                     # are any elements TRUE?
all(a>2)                                     # are all elements TRUE?

# From vectors to matrices
a <- matrix(1:25, nr=5, nc=5)                # create a matrix the "formal" way
a
a[1,2]                                       # select a matrix element (two dimensions)
a[1,]                                        # shortcut for ith row
all(a[1,]==a[1,1:5])                         # show the equivalence
a[,2]                                        # can also perform for columns
a[2:3,3:5]                                   # select submatrices
a[-1,]                                       # nice trick: negative numbers omit cells!
a[-2,-2]                                     # get rid of number two

b <- cbind(1:5,1:5)                          # another way to create matrices
b
d <- rbind(1:5,1:5)                          # can perform with rows, too
d
cbind(b,d)                                   # no go: must have compatible dimensions!
dim(b)                                       # what were those dimensions, anyway?
dim(d)
NROW(b)
NCOL(b)
cbind(b,b)                                   # here's a better example

t(b)                                         # can transpose b
cbind(t(b),d)                                # now it works
```

1.4. Element-wise operations

```
# Most arithmetic operators are applied element-wise
a <- 1:5
a + 1                                        # addition
a * 2                                        # multiplication
a / 3                                        # division
a - 4                                        # subtraction
a ^ 5                                        # you get the idea...

a + a                                        # also works on pairs of vectors
a * a
a %*% a                                      # note, not element-wise!
a + 1:6                                      # problem: need same length

a <- rbind(1:5,2:6)                          # same principles apply to matrices
b <- rbind(3:7,4:8)
a + b
a / b

a %*% t(b)                                   # matrix multiplication

# Logical operators (generally) work like arithmetic ones
a > 0
a == b
a != b
!(a == b)
(a>2) | (b>4)
(a>2) & (b>4)
(a>2) || (b>4)                               # beware the "double-pipe"!
(a>2) && (b>4)                               # (and the "double-ampersand"!)

# Ditto for many other basic transformations
log(a)
exp(b)
```

```
sqrt(a+b)                                # note that we can nest statements!
log((sqrt(a+b)+a)*b)                     # as recursive as we wanna be
```

1.5 Lists, data frames, and arrays

```
# R has many other data types.  One important type is the list.
a <- list(1:5)
a                                        # not an ordinary vector...
a <- list(1:5,letters[1:3])              # can we mix types and lengths?
a                                        # yes!
b <- matrix(1:3,3,3)
a <- list(1:5,letters[1:3],b)            # anything can be stuffed in here
a
a[[1]]                                   # retrieve the first item
a[[2]][3]                                # the letter "c"
(a[[3]])[1,3]                            # it's really just recursion again
a <- list(boo=1:4,hoo=5)                 # list elements are often named
names(a)                                 # get the element names
a[["boo"]]                               # ask for it by name
a$hoo                                    # use "$" to get what you want
a+3                                      # whoops - not a vector!
a[[1]]+3                                 # that works
a[[2]] <- a[[2]]*4                       # can also perform assignment
a$woo <- "glorp"                         # works with "$"
a[["foo"]] <- "shazam"                   # prolonging the magic
a

# Another useful generalization: the data frame
d <- data.frame(income=1:5,sane=c(T,T,T,T,F),name=LETTERS[1:5])  # Store multiple types
d
d[1,2]                                   # acts a lot like a matrix!
d[,1]*5
d[-1,]
names(d)                                 # also acts like a list
d[[2]]
d$sane[3]<-FALSE
d
d[2,3]                                   # hmm - our data got factorized!
d$name <- LETTERS[1:5]                   # eliminate evil factors by overwriting
d[2,3]
d <- data.frame(income=1:5,sane=c(T,T,T,T,F),name=LETTERS[1:5],stringsAsFactors=FALSE)
d                                        # another way to fix it

d <- as.data.frame(cbind(1:5,2:6))       # can create from matrices
d
is.data.frame(d)                         # how can we tell it's not a matrix?
is.matrix(d)                             # the truth comes out

# When two dimensions are not enough: arrays
a <- array(1:18, dim=c(2,3,3))           # now in 3D
a
a[1,2,3]                                 # selection works like a matrix
a[1,2,]
a[1,,]
a[-1,2:3,1:2]
a*5                                      # ditto for element-wise operations
a <- array(dim=c(2,3,2,5,6))             # can have any number of dimensions
dim(a)                                   # find out what we've allocated
```

1.6. Finding built-in data sets

```
# Many packages have built-in data for testing and educational purposes
data()                                   # list them all
data(package="base")                     # all base package
?USArrests                               # get help on a data set
data(USArrests)                          # load the data set
```

```
USArrests                                       # view the object
```

1.7. Elementary visualization

```
# R's workhorse is the "plot" command
plot(USArrests$Murder,USArrests$UrbanPop)
plot(USArrests$Murder,USArrests$UrbanPop,log="xy")    # log-log scale
plot(USArrests$Murder,USArrests$Assault,xlab="Murder",ylab="Assault",main="My Plot")

# Can also add text
plot(USArrests$Murder,USArrests$Assault,xlab="Murder",ylab="Assault",main="My Plot",type="n")
text(USArrests$Murder,USArrests$Assault,rownames(USArrests))

# Histograms and boxplots are often helpful
hist(USArrests$Murder)
boxplot(USArrests)
```

1.8. Reading in data (and writing to disk)

```
# We won't use them right now, but here are some useful commands:
?read.table                                           # a workhorse routine
?read.csv                                             # a specialized CSV version
?scan                                                 # a more low-level variant
apropos("read")                                       # list various "read" commands
?load                                                 # loads objects in native R format
?save                                                 # saves objects in native R format
?write.table                                          # counterpart to read.table
apropos("write")                                      # various "write" functions


# Remember that you can use the File menu to save your current global environment, change
# working directory, exit, etc.
```

APPENDIX 2: NETWORK OBJECTS: IMPORT, EXPLORATION, MANIPULATION

2.1 Built-in Datasets

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

#For more information....
?data
?flo
```

2.2 Importing Relational Data

```
# Be sure to be in the directory where you stored the data for the workshop
dir()                                   # Check what's in the working directory
#chwd("My File Location")               # Use if you need to change the working directory

#Read an adjacency matrix
floadj <- read.table("floadj.txt",header=TRUE)
floadj                                                         # Examine the matrix

#Read a Pajek file
flopaj <- read.paj("flo.paj")

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
nflo2                                # Examine the network object

#For more information....
?names
?read.paj
?read.table
```

2.3 Creating network Objects

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

#For more information....
?network
?as.network.matrix
```

2.4 Description and Visualization

```
summary(nflo)                                            # Get an overall summary
print(nflo)                                              # Simple print method
network.dyadcount(nflo)                                  # How many dyads in nflo?
network.edgecount(nflo)                                  # How many edges are present?
network.size(nflo)                                       # How large is the network?
as.sociomatrix(nflo)                                     # Show it as a sociomatrix
nflo[,]                                                  # Another way to do it

plot(nflo,displaylabels=T,boxed.labels=F)                # Plot with names
plot(nflo,displaylabels=T,mode="circle")                 # A less useful layout....
library(sna)                                             # Load the sna library
gplot(nflo)                                              # Requires sna

#For more information
?summary.network
```

```
?network.dyadcount
?network.edgecount
?as.sociomatrix
?as.matrix.network
?is.directed
?plot.network
```

2.5 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
m <- matrix(0, nrow=5, ncol=5)                          # Create an adjacency matrix
m[3,4:5] <- 1                                           # Add entries from 3 to 4 and 5
g[m>0] <- 1                                             # Add more entries
g

#Delete edges
g[3,5] <- 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
as.sociomatrix(nflo,attrname="boo")                     # Can also do it this way

#For more information....
?network.extraction
?add.edge
?delete.edges
?delete.vertices
?get.edges
```

2.6 Network and Vertex Attributes

```
#Add some attributes
nflo %v% "woo" <- letters[1:16]                         # Letter the families
nflo %n% "zoo" <- "R is TanFastic!"                     # A key network-level covariate

#Listing attributes
list.vertex.attributes(nflo)                            # List all vertex attributes
list.network.attributes(nflo)                           # List all network attributes

#Retrieving attributes
nflo %v% "woo"                                          # Retrieve the vertex attribute
nflo %n% "zoo"                                          # Retrieve the network attribute

#For more information
?attribute.methods
```

APPENDIX 3: CLASSICAL NETWORK ANALYSIS WITH SNA

3.1 Getting started

```
library(sna)                       # Load the sna library
help.start()                       # If not done already...walk through the various sna pages
library(help="sna")                # See also this for a list
load("sunbelt2011.Rdata")          # Load supplemental workshop data

#For more information....
?help.start
?library
?sna
```

3.2 Network data in `sna`

```
# sna can handle network data in many forms.  For instance, the function gden calculates
# network density; we can use it on a network object, an adjacency matrix, a list of
# such matrices, etc.
data(flo)
flo                                            # Adjacency form
gden(flo)
nflo<-network(flo,directed=FALSE)              # Network form
gden(nflo)
gden(list(flo,nflo))                           # Lists of matrices/networks
aflo<-array(dim=c(2,NROW(flo),NROW(flo)))      # Array form
aflo[1,,]<-flo
aflo[2,,]<-flo
gden(aflo)
gden(list(flo,aflo,nflo))                      # Yet more lists

# sna also supports a special kind of matrix called an "sna edgelist."  These are three-
# column matrices, each row of which represents an edge (via its sender, recipient, and
# value, respectively).  sna edgelists have special attributes that indicate their
# size, vertex names (if any), and bipartite status (if applicable).
eflo<-as.edgelist.sna(flo)                     # Coerce flo to an sna edgelist
eflo
attr(eflo,"n")                                 # How many vertices are there?
attr(eflo,"vnames")                            # Are there vertex names?
attr(eflo,"n")<-30                             # Could add isolates....
as.sociomatrix.sna(eflo)                       # Can transform back w/as.sociomatrix.sna

# sna edgelists can be handy with large data sets (as a simple alternative to network
# objects).  To make one, just add an "n" attribute to a valid three-column matrix!
mat<-cbind(rep(2,4),3:6,rep(1,4))              # Create edges from 2 to 3:6
attr(mat,"n")<-6                               # Set total number of vertices to 6
mat
gden(mat)                                      # Can now pass to sna routines
as.sociomatrix.sna(mat)                        # Can see in adjacency form

# For more information....
?as.edgelist.sna
?as.sociomatrix.sna
?attr
?sna
```

3.2 Network visualization with gplot

```
# Begin by plotting contiguity among nations in 1993 (from the Correlates of War project)
gplot(contig_1993)                             # The default visualization
gplot(contig_1993, usearrows=FALSE)            # Turn off arrows manually
gplot(contig_1993, gmode="graph")              # Can also tell gplot the data is undirected

# We can add labels to the vertices - network.vertex.names reports them
gplot(contig_1993, gmode="graph",
   label=network.vertex.names(contig_1993))

# This plot is too large/dense for the default settings to work.  Let's refine them.
gplot(contig_1993, gmode="graph", label.cex=0.5, label.col=4,
   label=network.vertex.names(contig_1993))     # Shrink labels and recolor

# Here's an example of directed data - militarized interstate disputes (MIDs) for 1993
gplot(mids_1993, label.cex=0.5, label.col=4,
   label=network.vertex.names(mids_1993))       # Basic display, with labels

# All those isolates can get in the way - we can suppress them using displayisolates
gplot(mids_1993, displayisolates=FALSE, label.cex=0.5, label.col=4,
   label=network.vertex.names(mids_1993))

# The default layout algorithm is that of Frutchterman-Reingold (1991), can use others
gplot(mids_1993, displayisolates=FALSE, label.cex=0.5, label.col=4,
   label=network.vertex.names(mids_1993), mode="circle")   # The infamous circle
gplot(mids_1993, displayisolates=FALSE, label.cex=0.5, label.col=4,
   label=network.vertex.names(mids_1993), mode="mds")      # MDS of position similarity

# When a layout is generated, the results can be saved for later reuse:
coords <- gplot(contig_1993)                     # Capture the magic of the moment
coords                                           # Show the vertex coordinates

#Saved (or a priori) layouts can be used via the coord argument:
gplot(mids_1993, label.cex=0.5, label.col=4, coord=coords,
   label=network.vertex.names(mids_1993))        # Relive the magic

# When the default settings are insufficient, interactive mode allows for tweaking
coords <- gplot(contig_1993, interactive=TRUE)   # Modify and save
gplot(contig_1993, coord=coords, gmode="graph")  # Should reproduce the modified layout

#For more information....
?gplot
?gplot.layout
```

3.3 Three-dimensional visualization with gplot3d (requires the rgl package)

```
gplot3d(contig_1993, label=network.vertex.names(contig_1993))   # Experience the future!
# Other layouts are possible here, too:
gplot3d(contig_1993, label=network.vertex.names(contig_1993), mode="kamadakawai")

#For more information....
?gplot3d
?gplot3d.layout
```

3.4 Basic centrality indices: degree, betweenness, and closeness

```
# We begin with the simplest case: degree
degree(mids_1993)                                              # Default: total degree
ideg <- degree(mids_1993, cmode="indegree")                    # Indegree for MIDs
odeg <- degree(mids_1993, cmode="outdegree")                   # Outdegree for MIDs
all(degree(mids_1993) == ideg+odeg)                            # In + out = total?

# Once centrality scores are computed, we can handle them using standard R methods:
plot(ideg, odeg, type="n", xlab="Incoming MIDs", ylab="Outgoing MIDs") # Plot ideg by odeg
abline(0, 1, lty=3)
text(jitter(ideg), jitter(odeg), network.vertex.names(contig_1993), cex=0.75, col=2)
#Plot simple histograms of the degree distribution:
par(mfrow=c(2,2))                                             # Set up a 2x2 display
hist(ideg, xlab="Indegree", main="Indegree Distribution", prob=TRUE)
hist(odeg, xlab="Outdegree", main="Outdegree Distribution", prob=TRUE)
hist(ideg+odeg, xlab="Total Degree", main="Total Degree Distribution", prob=TRUE)
par(mfrow=c(1,1))                                             # Restore display
```

```
# Centrality scores can also be used with other sna routines, e.g., gplot
gplot(mids_1993, vertex.cex=(ideg+odeg)^0.5/2, vertex.sides=50,
    label.cex=0.4, vertex.col=rgb(odeg/max(odeg),0,ideg/max(ideg)),
    label=network.vertex.names(mids_1993))

# Betweenness and closeness are also popular measures
bet <- betweenness(contig_1993, gmode="graph")              # Geographic betweenness
bet
gplot(contig_1993, vertex.cex=sqrt(bet)/25, gmode="graph")   # Use w/gplot
clo <- closeness(contig_1993)                                # Geographic closeness
clo                                                          # A large world after all?

# Can use sna routines to explore alternatives to the common measures....
closeness2 <- function(x){                         # Create an alternate closeness function!
    geo <- 1/geodist(x)$gdist                      # Get the matrix of 1/geodesic distance
    diag(geo) <- 0                                 # Define self-ties as 0
    apply(geo, 1, sum)                             # Return sum(1/geodist) for each vertex
}
clo2 <- closeness2(contig_1993)                  # Use our new function on contiguity data
hist(clo2, xlab="Alt. Closeness", prob=TRUE)     # Much better behaved!
cor(clo2, bet)                                   # Correlate with betweenness
plot(clo2, bet)                                  # Plot the bivariate relationship
all(clo2/185==closeness(contig_1993,cmode="suminvundir"))  # Actually, we support this in sna!

#For more information....
?betweenness
?bonpow
?closeness
?degree
?evcent
?graphcent
?infocent
?prestige
?stresscent
```

3.5 From centrality to centralization

```
centralization(mids_1993, degree, cmode="indegree")          # Do MIDs concentrate?
centralization(contig_1993, evcent)                          # Eigenvector centralization

#For more information....
?centralization
```

3.6 Elementary graph-level indices

```
gden(mids_1993)                                                 # Density
grecip(mids_1993)                                               # Dyadic reciprocity
grecip(mids_1993, measure="edgewise")                           # Edgewise reciprocity
grecip(mids_1993, measure="edgewise.lrr")                       # Reciprocation LRR
gtrans(mids_1993)                                               # Transitivity
log(gtrans(mids_1993)/gden(mids_1993))                          # Transitive completion LLR

#For more information....
?gden
?grecip
?gtrans
?hierarchy
```

3.7 Subgraph census routines

```
dyad.census(mids_1993)                                       # M,A,N counts
dyad.census(contig_1993)                                    # No As in undirected graphs
triad.census(mids_1993)                                     # Directed triad census
triad.census(contig_1993, mode="graph")                     # Undirected triad census
kpath.census(mids_1993, maxlen=6, tabulate.by.vertex=FALSE)   # Count paths of length <=6
kcycle.census(mids_1993, maxlen=6, tabulate.by.vertex=FALSE)  # Count cycles of length <=6
clique.census(mids_1993, tabulate.by.vertex=FALSE, enumerate=FALSE) # Find maximal cliques

#Can also look at more detailed tabulation/comembership for paths/cycles/cliques
kpath.census(mids_1993, maxlen=4)                           # Show tabulation by vertex
indirect <- kpath.census(mids_1993, maxlen=6, dyadic.tabulation="sum")$paths.bydyad
gplot(indirect, label.cex=0.4, vertex.cex=0.75, label=network.vertex.names(mids_1993))
                                                            # Plot indirect MIDs

#Component information can be obtained in various ways
components(mids_1993)                                     # Strong component count
components(mids_1993, connected="weak")                   # Weak component count
cd <- component.dist(mids_1993, connected="weak")         # Get weak components
cd
plot(1:length(cd$cdist), cd$cdist, xlab="Size", ylab="Frequency") # Component sizes
cl <- component.largest(mids_1993, connected="weak")      # Who's in the largest component?
cl
gplot(mids_1993[cl,cl], boxed.lab=FALSE, label.cex=0.5, label.col=4,
    label=network.vertex.names(mids_1993)[cl])            # Plot the largest weak component

#Likewise, many routines exist for handling isolates
is.isolate(mids_1993, 3)                            # Is the third vertex (BHM) an isolate?
is.isolate(mids_1993, "BHM")                        # The peaceful islands?
is.isolate(mids_1993, "USA")                        # Perhaps less so....
isol <- isolates(mids_1993)                         # Get the entire list of isolates
isol
network.vertex.names(mids_1993)[isol]               # Which countries are isolates?
gplot(mids_1993[-isol,-isol], label.cex=0.5, label.col=4,
    label=network.vertex.names(mids_1993)[-isol])   # Another way to remove isolates

#For more information....
?clique.census
?components
?component.dist
?dyad.census
?is.isolate
?isolates
?kcycle.census
?kpath.census
?triad.census
```

3.8 Elementary random graph generation

```
rgraph(10)                                         # A uniform random digraph of order 10
rgraph(10, tp=3/9)                                 # Homogeneous Bernoulli w/mean degree 3
rgraph(10, tp=3/9, mode="graph")                   # As above, but undirected
rgnm(1, 10, 20)                                    # Uniform conditional on order, edges
rguman(1, 10, mut=0.5, asym=0.25, null=0.25)       # Homogeneous multinomial on dyad census
rguman(1, 10, mut=0, asym=1, null=0)               # An extreme case: random tournament
gplot3d(rgws(1,50,1,2,0))                          # A Watts-Strogatz process - baseline
gplot3d(rgws(1,50,1,2,0.05))                       # ...with rewiring probability 0.05
gplot3d(rgws(1,50,1,2,0.2))                        # ...with rewiring probability 0.2

#For more information....
?rgbn
?rgmn
?rgraph
?rguman
?rgws
```

3.9 Basic connectivity/distance measurement, and cohesion

```
g <- rgraph(20, tp=3/19)                                # Start with a random digraph
g
is.connected(g)                                         # Is g strongly connected?
is.connected(g, connected="weak")                       # How about weakly connected?
geodist(g)                                              # Get information on geodesics
reachability(g)                                         # Return the reachability matrix
symmetrize(g)                                           # Symmetrize g using the "or" rule
symmetrize(g, rule="strong")                            # Symmetrize g using the "and" rule

#Several ways to get relatively cohesive groups
clique.census(g)                                        # Maximal clique census
kcores(g)                                               # k-cores (by degree)
bicomponent.dist(g)                                     # bicomponent properties
cutpoints(g)                                            # find articulation points

#Showing cohesion information can aid visualization
gplot(contig_1993,vertex.col=2+cutpoints(contig_1993,mode="graph",  # Show critical positions
    return.indicator=T))
kc<-kcores(contig_1993,cmode="indegree")                            # Show core nesting
gplot(contig_1993,vertex.col=rainbow(max(kc)+1)[kc+1])
gplot(contig_1993[kc>4,kc>4],vertex.col=rainbow(max(kc)+1)[kc[kc>4]+1]) # 5-core only

#For more information....
?bicomponent.dist
?cutpoints
?geodist
?kcores
?is.connected
?reachability
?symmetrize
```

3.10 Positional analysis

```
# Generate a structural equivalence clustering of the CoW alliance data
gplot(alliances_1993, gmode="graph", vertex.cex=0.5)              #An initial look....
ec <- equiv.clust(alliances_1993, mode="graph", plabels=network.vertex.names(alliances_1993))
ec                                                                # The clustering
plot(ec)                                                          # Plot the dendrogram
rect.hclust(ec$cluster, h=20)

# Use the clustering to form an SE blockmodel
bm <- blockmodel(alliances_1993, ec, h=20)
bm
#We can display the blockmodel in several ways....
plot.sociomatrix(alliances_1993[bm$order.vector,bm$order.vector], drawlab=FALSE)
bimage <- bm$block.model                                          # Extract the block image
bimage
bimage[is.nan(bimage)] <- 1
gplot(bimage, diag=TRUE, edge.lwd=bimage*5, vertex.cex=sqrt(table(bm$block.membership))/2,
    gmode="graph", vertex.sides=50, vertex.col=gray(1-diag(bimage)))  # Positional relations
```