APPENDIX A: R COMMANDS

```
##########################################################
#lines beginning with '#' are annotation                 #
#commands are noted in the text by the annotation number#
##########################################################

#Command 1: Install or update statnet

      install.packages( 'statnet' )

      update.packages( 'statnet' )

#Command 2: Load statnet
#this will need to be done every time R is opened to use statnet

      library( 'statnet' )

#Command 3: Accessing data available in R

      data()

#Command 4: Load the local health department R data

      data( lhds )

#Command 5: Check that the data loaded properly

      lhds

#Command 6: See a summary of the network and attributes

      summary( lhds )

#Command 7: Visualizing the network with nodes colored by state and HIV
#program attribute (Figure 3.1)
#dev.off( ) results in a refreshed graphic window with default settings for
#the new graphic
#palette assigns the default colors using in subsequent graphics

      dev.off( )

      par( mfrow = c( 2,1 ), mar = c( 0,0,1,0 ) )
      palette( gray.colors( 3,0,1 ) )
      plot( lhds, vertex.col = 'state', main = "State")
      plot( lhds, vertex.col = 'hivscreen', main = "HIV Screening Programs")
```

```
#Command 8: Visualize the largest component with nodes colored by HIV
#screening programming
#First syntax block identifies largest component, assigns to object, and
#saves object as network
#Second syntax block creates a subset of the HIV attribute vector and assigns
#to network object
#Third syntax block plots the new network with the HIV attribute vector,
#adding a legend (Figure 3.2)

      lhdscomp <- component.largest(lhds)
      lhdssmall <- lhds[lhdscomp,lhdscomp]
      smallnet<-as.network(lhdssmall, directed=FALSE)

      hivscreen <- get.vertex.attribute( lhds, 'hivscreen')
      subhiv<-as.vector(subset(hivscreen, lhdscomp!="FALSE"))
      smallnet %v% "hivscreen" <- subhiv

      dev.off( )
      par( mar = c( .2,.2,1,.2 ) )
      palette( gray.colors (2,0,1) )
      plot( smallnet, vertex.col = 'hivscreen', main = "HIV Screening
      Programs")
      legend("bottomleft", c("Y","N"),  pch=c(21,19))

#Command 9: Using node size to represent a continuous network measure
#R-Tip: Use the up and down arrows to scroll through syntax you've already
#used to reuse it

      numties <- degree( lhds, gmode='graph' )
      lhds %v% "numties" <- numties

      dev.off( )
      palette( gray.colors( 2,0,1 ) )

      par( mar = c( 0,0,0,0 ) )
      plot( lhds, vertex.col = 'hivscreen',vertex.cex = 'numties' )

#Command 10: Recoding degree and plotting the network (Figure 3.3)
#par allows editing of plot window features; mar is used to specify border
#size

      par( mar = c( 0,0,0,0 ) )
      plot( lhds, vertex.col = 'hivscreen',vertex.cex = numties/6 )

#Command 11: Degree and triangle distributions for the LHD network

      mean( degree( lhds, gmode = "graph" ) )
      sd( degree( lhds, gmode = "graph" ) )
      table( degree( lhds, gmode = "graph" ) )
```

```
      triad.census( lhds, mode = "graph" )

#Command 12: Create a random network the same size and density as LHD network
#set.seed allows simulations and statistical processes to be replicated
#exactly
#use goodness-of-fit procedures to get ESP and DSP values (Figure 3.4)

      set.seed( 2 )
      randomg <- rgraph( 1283, 1, tprob = .0033, mode = "graph" )
      randomnet <- as.network( randomg, directed = FALSE )

      summary( randomnet )

      nullrandom <- ergm( randomnet ~edges )
      gof_nullrandom <- gof( nullrandom, GOF =
      ~degree+distance+espartners+dspartners, seed = 567 )

      lhdforesp <- ergm( lhds ~edges )
      gof_lhdforesp <- gof( lhdforesp, GOF =
      ~degree+distance+espartners+dspartners, seed = 567 )

      dev.off( )

      par( mfrow = c( 3,2 ),mai = c( .8,.8,.2,.2 ) )

      hist( degree( lhds,gmode = "graph" ), breaks=max(degree(lhds))/2, main
      = "", xlab = "Degree ( LHD )",xlim = range( 0:15 ),ylim = range( 0:500
      ) )

      hist( degree( randomnet,gmode = "graph" ),
      breaks=max(degree(randomnet))/2, main = "", xlab = "Degree ( Random
      )",xlim = range( 0:15 ),ylim = range( 0:500 ) )

      barplot( gof_lhdforesp$obs.dspart[2:5], main = "", ylab = "Frequency",
      xlab = "DSP ( LHD )", xlim = range( 0:5 ), ylim = range( 0:15000 ) )

      barplot( gof_nullrandom$obs.dspart[2:5], main = "", xlab = "DSP (
      random network )", xlim = range( 0:5 ), ylim = range( 0:15000 ) )

      barplot( gof_lhdforesp$obs.espart[2:5], main = "", ylab =
      "Frequency",xlab = "ESP ( LHD )", xlim = range( 0:5 ),ylim = range(
      0:800 ) )

      barplot( gof_nullrandom$obs.espart[2:5], main = "", xlab = "ESP (
      random network )", xlim = range( 0:5 ),ylim = range( 0:800 ) )

#Command 13: Mixing matrices for HIV screening, nutrition programs, and years
#(Table 3.2)
```

```
mixingmatrix( lhds, "state" )
mixingmatrix( lhds, "hivscreen" )
mixingmatrix( lhds, "nutrition" )
mixingmatrix( lhds, "years" )
```

#Command 14: The number of links by experience level (Figure 3.5)

```
years <- get.vertex.attribute( lhds, 'years')
deg <- degree(lhds)

dev.off()
plot(tapply(deg, years, mean), type="o", axes = FALSE, ylab="Average
Connections", xlab="Leader Experience")
axis(1, at=1:4, lab=c("1-2 yrs", "3-5 yrs", "6-10 yrs", ">10 yrs"))
axis(2)
box()
```

#Command 15: Correlation between jurisdiction population and degree

```
popmil <- get.vertex.attribute(lhds, "popmil")
b <- data.frame(degree(lhds), popmil)
cor(b)
```

#Command 16: Other ways to explore attribute patterns

```
years <- get.vertex.attribute(lhds, "years")
nutrition <- get.vertex.attribute(lhds, "nutrition")
table( years, nutrition )
```

#Command 17: Estimating the LHD network null model (Table 3.3)

```
null <- ergm( lhds ~ edges )
summary( null )
```

#Command 18: Triangle distribution for simulated networks based on null model #(Figure 3.6)

```
simtrinull<-simulate( null, nsim = 100, monitor=~triangles,
statsonly=TRUE, control=control.simulate(MCMC.burnin=1000,
MCMC.interval=1000), seed=567)
lhds.tri <- summary( lhds~triangle )

dev.off()

par( mar = c( 4,4,1,1 ), cex.main = .9, cex.lab = .9,cex.axis = .75 )
hist(simtrinull[,2], xlim=c(0,1500), col='gray', main="", xlab="Number
of triangles", ylab="Number of simulations")
points(lhds.tri,3, pch="X", cex=2)
```

```
#Command 19: A main effects model including programming, population,
experience (Table 3.4)

      maineffects <- ergm( lhds ~ edges + nodecov( 'popmil' ) + nodefactor(
      'years' ) )
      summary( maineffects )

#Command 20: Changing the base argument for experience

      maineffects_base <- ergm( lhds ~ edges + nodecov( 'popmil' ) +
      nodefactor( 'years', base = 4 ) )
      summary( maineffects_base )

#Command 21: Exponentiating the coefficients for odds ratios and creating a
#table; for fewer significant digits, reduce the 4 following the digits =
#(Table 3.5)
#second set of commands is an alternative for obtaining test statistics
#note that second set of commands relies on the ste computed in the first set
#of commands
#Note that standard errors for dependence models are
#adjusted due to the use of MCMC estimation; this set
#of commands does not work for dependence models,
#use instructions in Appendix B instead

      or <- exp( maineffects$coef )
      ste <- sqrt( diag( maineffects$covar ) )
      lci <- exp( maineffects$coef-1.96*ste )
      uci <- exp( maineffects$coef+1.96*ste )
      oddsratios <- rbind( round( lci,digits = 4 ),round( or,digits = 4
      ),round( uci,digits = 4 ) )
      oddsratios <- t( oddsratios )
      colnames( oddsratios ) <- c( "Lower","OR","Upper" )
      oddsratios

      teststat <- maineffects$coef/ste
      teststats <- rbind( round( teststat,digits = 4 ))
      teststats <- t( teststats )
      colnames( teststats ) <- c( "Wald" )
      teststats

#Command 22: Listing R objects created during the estimation of the
#maineffects model
#use help( ergm ) for more information on each object

      names( maineffects )

#Command 23: Testing a model including homophily terms (Table 3.7)

      homophilymodel <- ergm( lhds ~ edges + nodecov( 'popmil' ) +
      nodefactor( 'years' ) + nodematch('hivscreen') + nodematch('nutrition')
      + nodematch('state') )
      summary( homophilymodel )
```

```
#Command 24: Modifying the homophily model to differential homophily for
#programming (Table 3.8)

      diffhomophily <- ergm( lhds ~ edges + nodecov( 'popmil' ) + nodefactor(
      'years' ) + nodematch('hivscreen', diff=T) + nodematch('nutrition',
      diff=T) + nodematch('state') )
      summary( diffhomophily )

#Command 25: Modifying the homophily model to keep only differential
#homophily for LHDs conducting the same programming (Table 3.9)

      diffhomophily2 <- ergm( lhds ~ edges + nodecov( 'popmil' ) +
      nodefactor( 'years' ) + nodematch('hivscreen', diff=T, keep=2) +
      nodematch('nutrition', diff=T, keep=2) + nodematch('state') )
      summary( diffhomophily2 )

#Command 26: Comparison of model fit for simulated networks from each model
#(Table 3.10)

      nullsim <- simulate(null, verbose = TRUE,seed = 5)
      mainsim <- simulate(maineffects, verbose = TRUE, seed = 5)
      homsim <- simulate(homophilymodel, verbose = TRUE, seed = 5)
      diffsim <- simulate(diffhomophily, verbose = TRUE, seed = 5)
      diff2sim <- simulate(diffhomophily2, verbose = TRUE, seed = 5)

      rowgof <- rbind(summary(lhds ~ edges + degree(0:5) + triangle),
            summary(nullsim ~ edges + degree(0:5) + triangle),
            summary(mainsim ~ edges + degree(0:5) + triangle),
            summary(homsim ~ edges + degree(0:5) + triangle),
            summary(diffsim ~ edges + degree(0:5) + triangle),
            summary(diff2sim ~ edges + degree(0:5) + triangle)
            )
      rownames(rowgof) <- c("lhds", "Null","Main effects","Homophily","Diff
      homophily","Diff homophily 2")
      rowgof

#Command 27: Network statistics from 10 simulated networks

      diff2.sim <- simulate(diffhomophily2,
      control=control.simulate(MCMC.burnin=1000, MCMC.interval=1000),nsim=10)
      summary(diff2.sim)

#Command 28: Observed network statistic for jurisdiction population

      summary(lhds ~ nodecov ("popmil"))

#Command 29: Simulating 100 networks based on the main effects model and
#comparing the number of triangles
```

```
simtri<-simulate( maineffects, nsim = 100, monitor=~triangles,
statsonly=TRUE,control=control.simulate(MCMC.burnin=1000,
MCMC.interval=1000), seed=1)
lhds.tri <- summary( lhds ~ triangle )
```

#Command 30: Simulating 100 networks based on the second differential #homophily model and comparing the number of triangles

```
simtri2 <- simulate( diffhomophily2, nsim = 100, monitor=~triangles,
statsonly=TRUE, control=control.simulate(MCMC.burnin=1000,
MCMC.interval=1000), seed=1)
```

#Command 31: Graphically comparing the number of triangles in the simulated #and observed networks (Figure 3.9)

```
par( mfrow = c( 1,2 ) )
hist( simtri[,6], main = "Main effects simulations",
xlim=c(0,1500),ylim=c(0,100), xlab = 'Number of triangles', ylab =
'Number of simulated networks' )
points(lhds.tri,4, pch="X", cex=2)
hist( simtri2[,9], main = "Diff Homophily 2 simulations",
xlim=c(0,1500),ylim=c(0,100),xlab = 'Number of triangles', ylab =
'Number of simulated networks')
points(lhds.tri,4, pch="X", cex=2)
```

#Command 32: Goodness-of-fit simulations for the diffhomophily2 model #(Table 3.11)

```
diff2_gof <- gof( diffhomophily2, GOF =  ~degree + espartners +
dspartners, verbose = T, burnin = 10000, interval = 10000, seed = 567 )
diff2_gof
```

#Command 33: See all values in goodness-of-fit tables for the diffhomophily2 #model

```
diff2_gof$pval.deg
diff2_gof$pval.espart
diff2_gof$pval.dspart
```

#Command 34: Graphic goodness-of-fit for the diffhomophily2 network #(Figure 3.10)
#the syntax for the second plot row shows how to change the y-axis to show #the log-odds of a node by adding plotlogodds = T

```
dev.off()
par( mfrow = c( 2,3 ) )
plot( diff2_gof, cex.lab = 1.5, cex.axis = 1.5 )
plot(diff2_gof, cex.lab = 1.5, cex.axis = 1.5, plotlogodds = T )
```

```
#Command 35: Selecting an alpha for the dependence model (Table 3.12)
#may want to remove unneeded objects and use gc() before running to clear
#memory for computationally intensive processes
#consider running one model at a time

      gwmodel1 <- ergm( lhds ~ edges + nodefactor( 'years' ) + nodecov(
      'popmil' ) + nodematch( 'hivscreen', diff=T, keep=2 ) + nodematch(
      'nutrition',diff = T, keep=2 ) + nodematch( 'state' ) + gwdegree( .1,T
      ) + gwesp( .1,T ) + gwdsp( .1,T ),
      control=control.ergm(MCMC.samplesize=100000, MCMC.burnin=1000000,
      MCMC.interval=1000, seed = 567), eval.loglik = T, verbose = T )
      summary( gwmodel1 )

      gwmodel2 <- ergm( lhds ~ edges + nodefactor( 'years' ) + nodecov(
      'popmil' ) + nodematch( 'hivscreen', diff=T, keep=2 ) + nodematch(
      'nutrition',diff = T, keep=2 ) + nodematch( 'state' ) + gwdegree( .2,T
      ) + gwesp( .2,T ) + gwdsp( .2,T ),
      control=control.ergm(MCMC.samplesize=100000, MCMC.burnin=1000000,
      MCMC.interval=1000, seed = 567), eval.loglik = T, verbose = T )
      summary( gwmodel2 )

      gwmodel3 <- ergm( lhds ~ edges + nodefactor( 'years' ) + nodecov(
      'popmil' ) + nodematch( 'hivscreen', diff=T, keep=2 ) + nodematch(
      'nutrition',diff = T, keep=2 ) + nodematch( 'state' ) + gwdegree( .3,T
      ) + gwesp( .3,T ) + gwdsp( .3,T ),
      control=control.ergm(MCMC.samplesize=100000, MCMC.burnin=1000000,
      MCMC.interval=1000, seed = 567), eval.loglik = T, verbose = T )
      summary( gwmodel3 )

      gwmodel4 <- ergm( lhds ~ edges + nodefactor( 'years' ) + nodecov(
      'popmil' ) + nodematch( 'hivscreen', diff=T, keep=2 ) + nodematch(
      'nutrition',diff = T, keep=2 ) + nodematch( 'state' ) + gwdegree( .4,T
      ) + gwesp( .4,T ) + gwdsp( .4,T ),
      control=control.ergm(MCMC.samplesize=100000, MCMC.burnin=1000000,
      MCMC.interval=1000, seed = 567), eval.loglik = T, verbose = T )
      summary( gwmodel4 )

      gwmodel5 <- ergm( lhds ~ edges + nodefactor( 'years' ) + nodecov(
      'popmil' ) + nodematch( 'hivscreen', diff=T, keep=2 ) + nodematch(
      'nutrition',diff = T, keep=2 ) + nodematch( 'state' ) + gwdegree( .5,T
      ) + gwesp( .5,T ) + gwdsp( .5,T ),
      control=control.ergm(MCMC.samplesize=100000, MCMC.burnin=1000000,
      MCMC.interval=1000, seed = 567), eval.loglik = T, verbose = T )
      summary( gwmodel5 )

      gwmodel6 <- ergm( lhds ~ edges + nodefactor( 'years' ) + nodecov(
      'popmil' ) + nodematch( 'hivscreen', diff=T, keep=2 ) + nodematch(
      'nutrition',diff = T, keep=2 ) + nodematch( 'state' ) + gwdegree( .6,T
```

```
) + gwesp( .6,T ) + gwdsp( .6,T ),
control=control.ergm(MCMC.samplesize=100000, MCMC.burnin=1000000,
MCMC.interval=1000, seed = 567), eval.loglik = T, verbose = T )
summary( gwmodel6 )

gwmodel7 <- ergm( lhds ~ edges + nodefactor( 'years' ) + nodecov(
'popmil' ) + nodematch( 'hivscreen', diff=T, keep=2 )
+ nodematch( 'nutrition',diff = T, keep=2 ) + nodematch( 'state' ) +
gwdegree( .7,T ) + gwesp( .7,T ) + gwdsp( .7,T ),
control=control.ergm(MCMC.samplesize=100000, MCMC.burnin=1000000,
MCMC.interval=1000, seed = 567), eval.loglik = T, verbose = T )
summary( gwmodel7 )

gwmodel10 <- ergm( lhds ~ edges + nodefactor( 'years' ) + nodecov(
'popmil' ) + nodematch( 'hivscreen', diff=T, keep=2 )
+ nodematch( 'nutrition',diff = T, keep=2 ) + nodematch( 'state' ) +
gwdegree( 1,T ) + gwesp( 1,T ) + gwdsp( 1,T ),
control=control.ergm(MCMC.samplesize=100000, MCMC.burnin=1000000,
MCMC.interval=1000, seed = 567), eval.loglik = T, verbose = T )
summary( gwmodel10 )

gwmodel11 <- ergm( lhds ~ edges + nodefactor( 'years' ) + nodecov(
'popmil' ) + nodematch( 'hivscreen', diff=T, keep=2 )
+ nodematch( 'nutrition',diff = T, keep=2 ) + nodematch( 'state' ) +
gwdegree( 1.1,T ) + gwesp( 1.1,T ) + gwdsp( 1.1,T ),
control=control.ergm(MCMC.samplesize=100000, MCMC.burnin=1000000,
MCMC.interval=1000, seed = 567), eval.loglik = T, verbose = T )
summary( gwmodel11 )
```

#Command 36: MCMC diagnostics for dependence model alpha = 1 (Figure 3.11)
#to see all diagnostic graphs, turn on recording in the History menu for the
#graphics window before running Syntax, use page up and page down to scroll
#through resulting graphics

```
gc()
mcmc.diagnostics( gwmodel10 )
```

#Command 37: Curved exponential family model (Table 3.13) and diagnostics

```
cefmodel <- ergm( lhds ~ edges + nodefactor( 'years' ) + nodecov(
'popmil' ) + nodematch( 'hivscreen', diff=T, keep=2 )
+ nodematch( 'nutrition',diff = T, keep=2 ) + nodematch( 'state' ) +
gwdegree( 1,F ) + gwesp( 1,F ) + gwdsp( 1,F ),
control=control.ergm(MCMC.samplesize=100000, MCMC.burnin=1000000,
MCMC.interval=1000, seed = 567), eval.loglik = T, verbose = T  )
summary( cefmodel )

mcmc.diagnostics( cefmodel )
```

```
#Command 38: Comparison of network measures across all six estimated models
#and the LHD network (Table 3.14)

      nullsim <- simulate(null, verbose = TRUE, seed = 5)
      mainsim <- simulate(maineffects, verbose = TRUE, seed = 5)
      homsim <- simulate(homophilymodel, verbose = TRUE, seed = 5)
      diffsim <- simulate(diffhomophily, verbose = TRUE, seed = 5)
      diffsim2 <- simulate(diffhomophily2, verbose = TRUE, seed = 5)
      gwsim <- simulate(gwmodel10, verbose = TRUE, seed = 5)
      cefsim <- simulate(cefmodel, verbose = TRUE, seed = 5)

      rowgof <- rbind(summary(lhds ~ edges + degree(0:5) + triangle),
            summary(nullsim ~ edges + degree(0:5) + triangle),
            summary(mainsim ~ edges + degree(0:5) + triangle),
            summary(homsim ~ edges + degree(0:5) + triangle),
            summary(diffsim ~ edges + degree(0:5) + triangle),
            summary(diffsim2 ~ edges + degree(0:5) + triangle),
            summary(gwsim ~ edges + degree(0:5) + triangle),
            summary(cefsim ~ edges + degree(0:5) + triangle)
            )
      rownames(rowgof) <- c("LHD", "Null","Main effects","Homophily","Diff
      homophily", "Diff homophily 2", "Dependence", "CEF model")
      rowgof

#Command 39: Distribution of triangles in differential homophily and CEF
#models (Figure 3.12)

      simtri0 <- simulate( diffhomophily2, nsim = 100, monitor=~triangles,
      statsonly=TRUE, control=control.simulate(MCMC.burnin=1000,
      MCMC.interval=1000), seed=1)

      simtri1 <- simulate( gwmodel10, nsim = 100, monitor=~triangles,
      statsonly=TRUE,control=control.simulate(MCMC.burnin=1000,
      MCMC.interval=1000), seed=1)

      simtri2 <- simulate( cefmodel, nsim = 100, monitor=~triangles,
      statsonly=TRUE,control=control.simulate(MCMC.burnin=1000,
      MCMC.interval=1000), seed=1)

      dev.off()

      par( mfrow = c( 1,3 ), mar = c( 4,4,2.5,1 ), cex.main = .9, cex.lab =
      .9,cex.axis = .75 )

      hist( simtri0[,9],main = "Diff homophily 2 model", xlim =
      range(500:1500), ylim = range(0:35), xlab = 'Number of triangles', ylab
      = 'Number of simulations' )
      points(lhds.tri,4, pch="X", cex=2)
```

```
hist( simtri1[,12],main = "Dependence model", xlim = range(1300:1500),
ylim = range(0:35), xlab = 'Number of triangles', ylab = 'Number of
simulations' )
points(lhds.tri,4, pch="X", cex=2)

hist( simtri2[,99], main = "CEF model", xlim = range(1300:1500), ylim =
range(0:35), xlab = 'Number of triangles',ylab = 'Number of
simulations' )
points(lhds.tri,4, pch="X", cex=2)
```

#Command 40: Other goodness-of-fit plots for the dependence and CEF models #(Figure 3.13)

```
gwmodel_gof <- gof( gwmodel10, GOF = ~ degree + espartners +
dspartners, burnin = 1000000, interval = 1000, seed = 567 )

cefmodel_gof <- gof( cefmodel, GOF = ~ degree + espartners +
dspartners, burnin = 1000000, interval = 1000, seed = 567 )

dev.off( )

par( mfrow = c( 2,3 ), mar = c( 4,4,4,1 ), cex.main = .9, cex.lab = .9,
cex.axis = .75 )

plot( gwmodel_gof, cex.lab = 1.5, cex.axis = 1.5, plotlogodds = T )
plot( cefmodel_gof, cex.lab = 1.5, cex.axis = 1.5, plotlogodds = T )
```

#Command 41: Graphic showing four networks based on LHD and simulations of #models (Figure 3.14)

```
dev.off()
palette( gray.colors( 2,0,1 ) )
par(mfrow=c(2,2),mar=c(.1,.1,1.2,.1))
plot(lhds, vertex.col="hivscreen", main="LHD")
plot(mainsim, vertex.col="hivscreen", main="Main effects")
plot(homsim, vertex.col="hivscreen", main="Homophily")
plot(cefsim, vertex.col="hivscreen", main="CEF")
```

#Command 42: Bringing in the Lake Pomona directed network data

```
data( emon )
lake <- emon$LakePomona

summary( lake )
```

#Command 43: Creating a random network of the same size & density as Lake #Pomona network and comparing visually (Figure 4.1)

```
set.seed( 567 )
```

```
randomdir <- rgraph( 20, 1, tprob = .39, mode = "digraph" )
randomdirnet <- as.network( randomdir, directed = TRUE )

par( mfrow = c( 1,2 ), mai = c( 0,0,.5,0 ) )
plot( lake, main = "Lake Pomona",vertex.cex = 2, edge.col = "grey40",
vertex.col = 200, displayisolates = F, mode = "kamadakawai" )

plot( randomdirnet, main = "Random network", edge.col = "grey40",
vertex.cex = 2, vertex.col = 200,displayisolates = F,mode =
"kamadakawai" )
```

#Command 44: Histograms of indegree and outdegree (Figure 4.2)

```
dev.off( )

par( mfrow = c( 2,2 ), mai = c( .75,.75,.2,.2 ) )

hist( degree( lake, cmode = "indegree" ), main = "",
breaks=max(degree(lake, cmode="indegree"))/2, xlab = "Indegree ( Lake
Pomona )", xlim = range( 0:20 ), ylim = range( 0:8 ) )

hist( degree( randomdirnet,cmode = "indegree" ), main = "",
breaks=max(degree(randomdirnet, cmode="indegree"))/2, xlab = "Indegree
( random network )", xlim = range( 0:20 ), ylim = range( 0:8 ) )

hist( degree( lake,cmode = "outdegree" ), main = "",
breaks=max(degree(lake, cmode="outdegree"))/2, xlab = "Outdegree ( Lake
Pomona )", xlim = range( 0:20 ), ylim = range( 0:8 ) )

hist( degree( randomdirnet,cmode = "outdegree" ), main = "",
breaks=max(degree(randomdirnet, cmode="outdegree"))/2, xlab =
"Outdegree ( random network )", xlim = range( 0:20 ), ylim = range( 0:8
) )
```

#Command 45: Bar graphs of dyad types (Figure 4.3)

```
par( mfrow = c( 1,2 ), mai = c( .5,1,.5,.5 ) )

barplot( dyad.census( lake ), xlab = "Dyad types", names.arg = c(
"Mutual", "Asymmetrical", "Null" ), main = "Lake Pomona" )

barplot( dyad.census( randomdirnet ),xlab = "Dyad types", names.arg =
c( "Mutual", "Asymmetrical", "Null" ), main = "Random network" )
```

#Command 46: Bar graph of triad census (Figure 4.4)

```
dev.off( )

laketri <- triad.census( lake )
```

```
randtri <- triad.census( randomdirnet )
triads <- rbind( laketri,randtri )
triads

barplot( as.matrix( triads ),beside = TRUE,col = c( 225,200 ) )
legend( "topright", inset = .05, title = "Triad Census", c( "Lake
Pomona","Random" ), horiz = FALSE, pch = 15, col = c( 225,200 ) )
```

#Command 47: Null and p1 models for Lake Pomona network

```
nullpomona <- ergm ( lake ~ edges )
summary( nullpomona )

p1model <- ergm ( lake ~ edges + sender + receiver + mutual,
control=control.ergm(MCMC.samplesize=100000, MCMC.burnin=1000000,
MCMC.interval=1000, seed = 567), eval.loglik = T, verbose = T )

summary( p1model )
```

#Command 48: Adding base arguments to the p1 model for the Lake Pomona network

```
p1model2 <- ergm ( lake ~ edges + sender + receiver(base=c(1,5,8,19)) +
mutual, control=control.ergm(MCMC.samplesize=100000,
MCMC.burnin=1000000, MCMC.interval=1000, seed = 567), eval.loglik = T,
verbose = T )
summary( p1model2 )
```

#Command 49: Simulations of mutual dyads from null and p1 Pomona models #(Figure 4.5)

```
simnullpomona<-simulate( nullpomona,nsim = 100, monitor=~mutual,
statsonly=TRUE,control=control.simulate(MCMC.burnin=1000000,
MCMC.interval=1000), seed=1)
lake.mutual<-summary( lake ~ mutual )

simp1<-simulate( p1model2, nsim = 100, monitor=~mutual, statsonly=TRUE,
control=control.simulate(MCMC.burnin=1000000, MCMC.interval=1000),
seed=1)
lake.mutual<-summary( lake ~ mutual )

dev.off( )

par( mfrow = c( 1,2 ), mai = c( .5,1,.5,.2 ) )

hist( simnullpomona[,2], main = "Null model",breaks=max(degree(lake,
cmode="indegree"))/4, xlim = range(0:60),ylim = range(0:45),xlab =
'Number of mutual dyads', ylab = 'Number simulated networks' )
```

```
points( lake.mutual, 4, pch="X", cex=2 )

hist( simp1[,37], main = "p1 model",breaks=max(degree(lake,
cmode="indegree"))/4, xlim = range(0:60),ylim = range(0:45),xlab =
'Number of mutual dyads', ylab = 'Number simulated networks' )

points( lake.mutual, 4, pch="X", cex=2 )
```

#Command 50: Graphic and mixing matrix examining local and nonlocal #organizations in Pomona network (Figure 4.6; Table 4.2)

```
dev.off( )
palette( gray.colors( 2,0,1 ) )
par( mai = c( .2,.2,0,0 ) )

plot( lake, vertex.col = 'Location', vertex.cex = 2 )
legend( "bottomleft",c( "Local","Non-local","Both" ),pt.bg = c(
"black","white","gray" ),pch = 21 )

mixingmatrix( lake, 'Location' )
```

#Command 51: Main effects Pomona model with location as a factor #(Tables 4.3, 4.4)

```
lakelocation <- ergm( lake ~ edges + nodeifactor( 'Location' ) )
summary( lakelocation )

lakelocation2 <- ergm( lake ~ edges + nodeifactor( 'Location' ) +
gwdsp( .1, T )+gwesp( .1, T ),
control=control.ergm(MCMC.samplesize=100000, MCMC.burnin=1000000,
MCMC.interval=1000, seed = 567), eval.loglik = T, verbose = T )
summary( lakelocation2 )
```

#Command 52: Goodness-of-fit for main effects and dependence models #(Figure 4.7)

```
lakelocation_gof <- gof( lakelocation, GOF = ~odegree + idegree +
espartners + dspartners, burnin = 1000000, interval = 1000, seed = 567
)
lakelocation_gof

dev.off( )

par( mfrow = c( 2,2 ),mai = c( .75,1,.1,.2 ) )

plot( lakelocation_gof, cex.lab = 1.5, cex.axis = 1.5, main = "" )
```

```
lakelocation2_gof <- gof( lakelocation2, GOF = ~odegree + idegree +
espartners + dspartners, burnin = 1000000, interval = 1000, verbose=T,
seed = 567 )
lakelocation2_gof

dev.off( )

par( mfrow = c( 2,2 ),mai = c( .75,1,.1,.2 ) )

plot( lakelocation2_gof, cex.lab = 1.5, cex.axis = 1.5, main = "" )
```

#Command 53: Bringing in Coleman data for using a network as a predictor
#network plots for fall and spring networks (Figure 4.9; Table 4.5)

```
data( coleman )
coleman
i = 1:73
fall <- rbind( coleman[1,i,] )
spring <- rbind( coleman[2,i,] )
spring <- as.network( spring )
spring %e% "fall" <- fall

fall2 <- as.network( fall )

par( mfrow = c( 1,2 ), mar = c( 0,0,1.5,0 ) )

plot( fall2, vertex.col = 'white', displayisolates = FALSE, main =
"Fall", vertex.cex = 2 )

plot( spring, vertex.col = 'grey', displayisolates = FALSE, main =
"Spring", vertex.cex = 2 )
```

#Command 54: Coleman null and main effects models (Table 4.6)

```
nullcoleman <- ergm ( spring ~ edges )
summary( nullcoleman )

maineffcoleman <- ergm( spring ~ edges + edgecov( fall ) )
summary( maineffcoleman )
```

APPENDIX B: MODIFYING R-ERGM MODEL SUMMARY PROCEDURE USING FIX()

Additions and deletions can be made to the summary table and any other of the ergm objects in R-statnet; however, be aware that any adjustments made will persist until the statnet or ergm package is reinstalled or the change is reversed manually. In addition, if the package is reinstalled, any adjustments made to the underlying code will be lost. It is important, therefore, to save any code modifications for use with future versions of statnet and ergm so they are not lost.

(1) Including the Wald test statistic

Open the ergm summary code using the fix syntax: fix(summary.ergm)

To include the Wald statistic in your summary table, scroll through the code to the rows that look like this:

```
tval <- object$coef/asyse
pval <- 2 * pt(q = abs(tval), df = rdf, lower.tail = FALSE)
count <- 1
templist <- NULL
while (count <= length(names(object$coef))) {
templist <- append(templist, c(object$coef[count], asyse[count],
            object$mc.se[count], pval[count]))
        count <- count + 1
    }
tempmatrix <- matrix(templist, ncol = 4, byrow = TRUE)
colnames(tempmatrix) <- c("Estimate", "Std. Error", "MCMC s.e.",
        "p-value")
```

Add the test statistic value (tval) to the output table (shaded). Be sure to change the number of columns to 5 rather than 4:

```
tval <- object$coef/asyse
pval <- 2 * pt(q = abs(tval), df = rdf, lower.tail = FALSE)
count <- 1
templist <- NULL
    while (count <= length(names(object$coef))) {
        templist <- append(templist, c(object$coef[count],
asyse[count],
            object$mc.se[count],tval[count], pval[count]))
        count <- count + 1
    }
tempmatrix <- matrix(templist, ncol = 5, byrow = TRUE)
colnames(tempmatrix) <- c("Estimate", "Std. Error", "MCMC s.e.",
"Wald",
 "p-value")
```

Save and close the fix window.

(2) Adding odds ratios and confidence intervals to the ergm summary table

To add odds ratios and a 95% confidence interval to your summary table for all subsequent models, use the fix(summary.ergm) command at the prompt. A window will open showing the underlying code processed when the summary function is called. Scroll through the code until you find the rows that look like this:

```
pval <- 2 * pt(q = abs(tval), df = rdf, lower.tail = FALSE)
 if (any(!is.na(mc.se))) {
      mc.se[!is.na(mc.se)] <- round(100 * mc.se[!is.na(mc.se)] *
            mc.se[!is.na(mc.se)]/(asyse[!is.na(mc.se)] *
            asyse[!is.na(mc.se)]))
 }
 count <- 1
 templist <- NULL
 while (count <= length(names(object$coef))) {
     templist <- append(templist, c(object$coef[count], asyse[count],
         mc.se[count], pval[count]))
     count <- count + 1
 }
 tempmatrix <- matrix(templist, ncol = 4, byrow = TRUE)
 colnames(tempmatrix) <- c("Estimate", "Std. Error", "MCMC %",
     "p-value")
```

Modify the code to include the calculations and display of odds ratios and confidence intervals (see highlighted additions in the code below):

```
pval <- 2 * pt(q = abs(tval), df = rdf, lower.tail = FALSE)
or <- exp(object$coef) #calculates odds ratios for model specified in
summary
lci <- exp(object$coef - 1.96*asyse) #calculates lower 95% CI
uci <- exp(object$coef + 1.96*asyse) #calculates upper 95% CI
if (any(!is.na(mc.se))) {
        mc.se[!is.na(mc.se)] <- round(100 * mc.se[!is.na(mc.se)] *
            mc.se[!is.na(mc.se)]/(asyse[!is.na(mc.se)] *
asyse[!is.na(mc.se)]))
    }
count <- 1
templist <- NULL
while (count <= length(names(object$coef))) {
        templist <- append(templist, c(object$coef[count],
asyse[count],
            mc.se[count], lci[count],or[count],uci[count],
pval[count]))
        count <- count + 1
    }
tempmatrix <- matrix(templist, ncol = 7, byrow = TRUE)
colnames(tempmatrix) <- c("Estimate", "Std. Error", "MCMC %","Lower",
"OR", "Upper", "p-value")
```

Using this modified summary code, the summary table for the main effects model now includes columns showing the odds ratio and 95% confidence interval for each term in the model. Note that this table reinstates the original two-sided *p* value calculation and removed the *t* statistic.