Young ABM Example

Bernard A. Coles IV

9/4/2019

This lab examines longitudinal network structures using the `RSiena` *package. Information about the Siena program can be found at:* http://www.stats.ox.ac.uk/~snijders/siena/

Getting Started

As discussed in the Introduction to Stochastic Actor-Based Models lecture, we are trying to build a model that accurately represents the preferences among actors that generated the observed network between discrete time points.

Let's start by working with the advice network among 75 MBA students measured at three waves. At each wave, the students were asked to whom they ask for advice. Keep in mind, as we look through the advice network over three time periods, that we are trying to model the decisions actors make about the ties they send.

```r
# clear the workspace and load the libraries.
rm(list = ls())
library(sna)
library(network)

# Load the data and assign each network to an object.
advice1 <- as.matrix(read.csv("https://www.jacobtnyoung.com/uploads/2/3/4/5/23459640/mba-advice1.csv",a
advice2 <- as.matrix(read.csv("https://www.jacobtnyoung.com/uploads/2/3/4/5/23459640/mba-advice2.csv",a
advice3 <- as.matrix(read.csv("https://www.jacobtnyoung.com/uploads/2/3/4/5/23459640/mba-advice3.csv",a

# Now, coerce them to be a network object.
advice1.net <- as.network(advice1)
advice2.net <- as.network(advice2)
advice3.net <- as.network(advice3)

# take a look at the properties of each network.
summary(advice1.net, print.adj = FALSE)
```

```
## Network attributes:
##   vertices = 75
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##  total edges = 307
##    missing edges = 0
##    non-missing edges = 307
##  density = 0.05531532
##
```

```
## Vertex attributes:
##   vertex.names:
##    character valued attribute
##    75 valid vertex names
##
## No edge attributes
```

```
summary(advice2.net, print.adj = FALSE)
```

```
## Network attributes:
##   vertices = 75
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##  total edges = 367
##    missing edges = 0
##    non-missing edges = 367
##  density = 0.06612613
##
## Vertex attributes:
##   vertex.names:
##    character valued attribute
##    75 valid vertex names
##
## No edge attributes
```

```
summary(advice3.net, print.adj = FALSE)
```

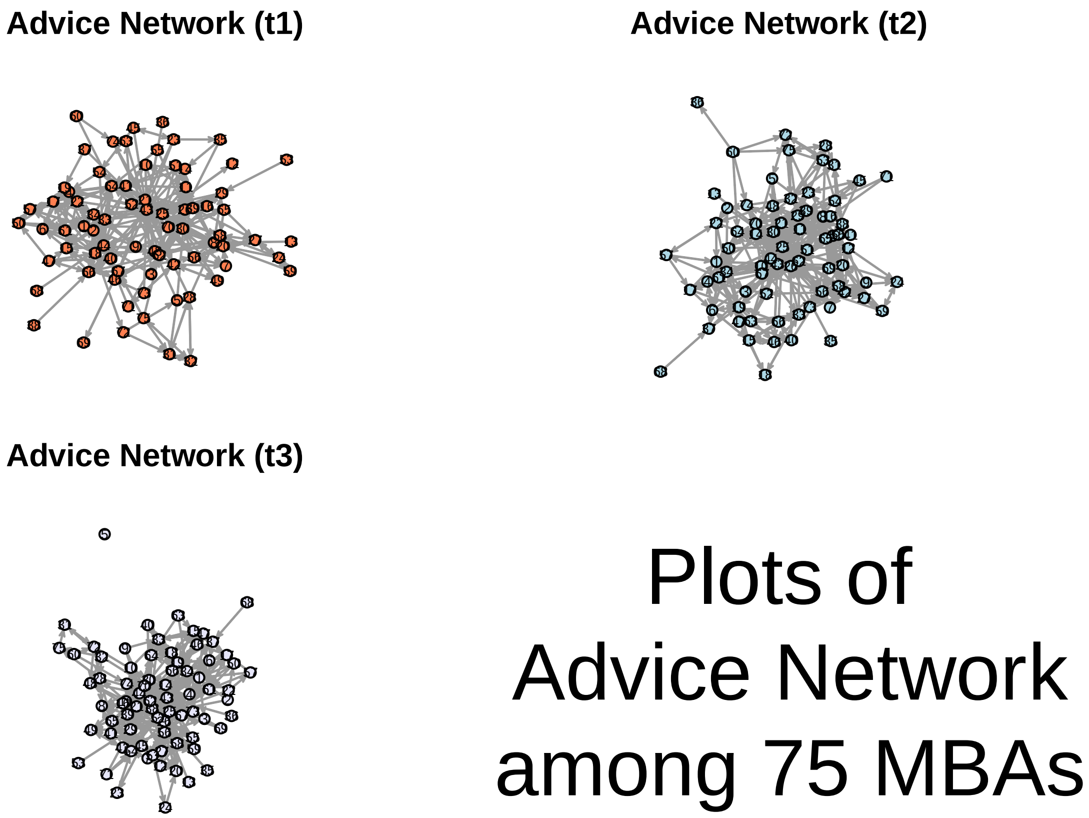
```
## Network attributes:
##   vertices = 75
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##  total edges = 341
##    missing edges = 0
##    non-missing edges = 341
##  density = 0.06144144
##
## Vertex attributes:
##   vertex.names:
##    character valued attribute
##    75 valid vertex names
##
## No edge attributes
```

What do the summaries tell us about each network?

Let's take a look at the plots of each network to see what patterns we can discern over time.

```
op <- par(mfrow=c(2,2), mai = c(0,0,0.7,0))
set.seed(605)
```

```
gplot(advice1.net, edge.col = "grey60", vertex.col="coral",label = seq(1,75), vertex.cex=1.5, label.pos
title("Advice Network (t1)")
set.seed(605)
gplot(advice2.net, edge.col = "grey60", vertex.col="lightblue",label = seq(1,75), vertex.cex=1.5, label
title("Advice Network (t2)")
gplot(advice3.net, edge.col = "grey60", vertex.col="lavender",label = seq(1,75), vertex.cex=1.5, label.
title("Advice Network (t3)")
plot(c(0, 1), c(0, 1), ann = F, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
text(x = 0.5, y = 0.5, paste("Plots of\n Advice Network\n among 75 MBAs"),cex = 3, col = "black")
```

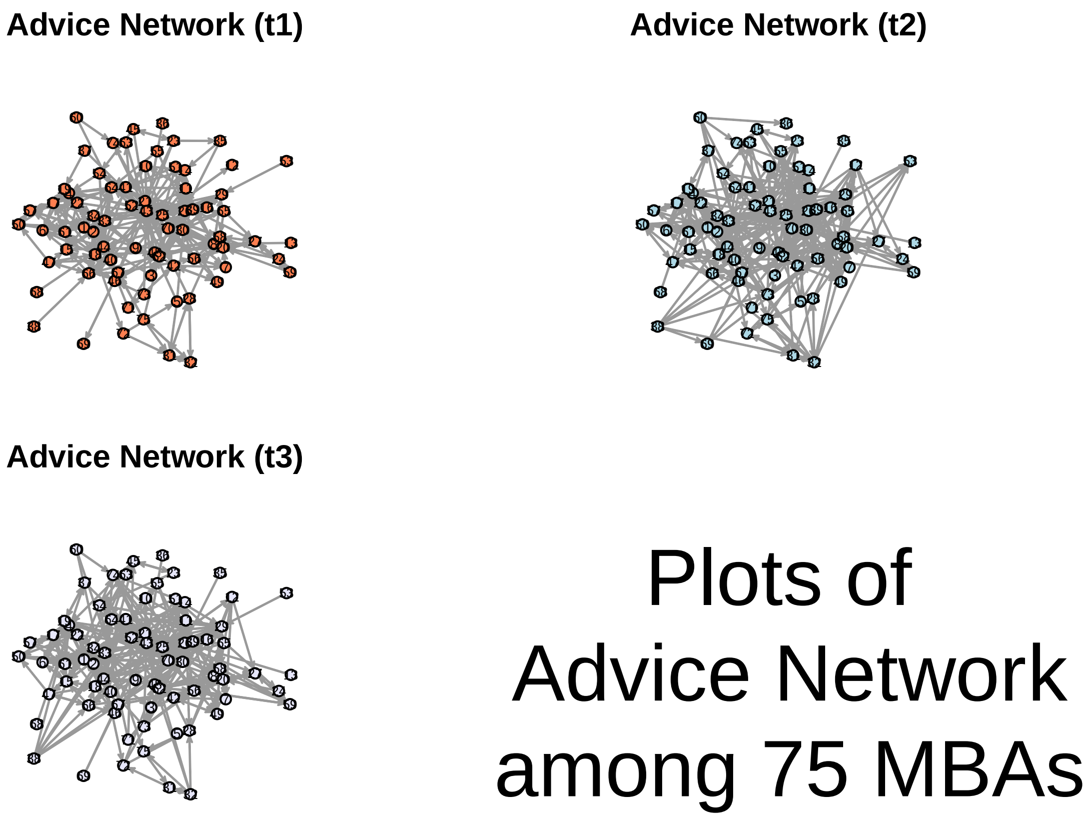



```
par(op)
```

What do the plots show us about the evolution of the network over time?

Let's force the individuals to "stay" in the same position in the plot and see how individual actors' indegree and outdegree change over time.

```
op <- par(mfrow=c(2,2), mai = c(0,0,0.7,0))
set.seed(605)
coords <- gplot(advice1.net, edge.col = "grey60", vertex.col="coral",label = seq(1,75), vertex.cex=1.5,
title("Advice Network (t1)")
set.seed(605)
gplot(advice2.net, edge.col = "grey60", vertex.col="lightblue", coord = coords,label = seq(1,75), verte
title("Advice Network (t2)")
gplot(advice3.net, edge.col = "grey60", vertex.col="lavender", coord = coords,label = seq(1,75), vertex
title("Advice Network (t3)")
plot(c(0, 1), c(0, 1), ann = F, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
text(x = 0.5, y = 0.5, paste("Plots of\n Advice Network\n among 75 MBAs"),cex = 3, col = "black")
```




```r
par(op)
```

Take a look at a few specific nodes. What do the plots show us about the evolution of the network over time?

Now, we want to think about modeling the evolution of this network over time. To do that, we need to look through the `RSiena` package.

The `RSiena` package

The `RSiena` package performs simulation-based estimation of stochastic actor-based models for longitudinal network data collected as panel data. Dependent variables can be single or multivariate networks, which can be directed, non-directed, or two-mode. There are also functions for testing parameters and checking goodness of fit.

If you have not already done so, install the package using `install.packages("RSiena")`. If you have not installed the `RSiena` package recently, make sure you update using the `update.packages("RSiena")` command.

Note that if you are using the MacOS, that `RSiena` will require the XQuartz software. Please download the XQuartz software at: https://www.xquartz.org/ before running `RSiena`.

```r
# Load the RSiena package.
library(RSiena)

# Take a look at the help.
help(package="RSiena")
```

For an RSiena analysis, we have to build several objects. The objects will serve as variables:

- A data object to examine (using `sienaDependent()` and `sienaDataCreate()`).
- A set of effects to estimate (using `getEffects` and `includeEffects()`).
- A model object that will have the terms we want to estimate (using `sienaAlgorithmCreate` or `sienaModelCreate()`).
- Then, we estimate the model using `siena07()`.

Step 1: Building the object to analyze using the `sienaDependent()` and `sienaDataCreate()` functions

First, we want to create an object that is an array of the networks that will be the **dependent** variable. To do this, we use the `sienaDependent()` function. To see how this function works, look at the help with `?sienaDependent`.

In our example, we will create an object called `advice` which is the three networks constructed as an `array`. The `array` will have dimensions *75 x 75 x 3*. That is, the number of individuals in each network represented over three time points. If, for example, we had a network with fifty individuals and four waves of data, then our array would be *50 x 50 x 4*. Think of the object we have created as a cube that has 75 rows and 75 columns and each slice of the cube is a cross-section of the network.

```
# Take a look at the help.
?sienaDependent

# Build the network object to examine.
advice <- sienaDependent(
  array(                        # define that it is an array.
    c(advice1,advice2,advice3), # define the three networks.
    dim=c(75,75,3)              # 75 x 75 x 3 are the dimensions.
    )
  )

# we see that advice is a one mode network, with 3 observations, and 75 actors.
advice
```

```
## Type         oneMode
## Observations 3
## Nodeset      Actors (75 elements)
```

```
# advice is an object of class "sienaDependent"
class(advice)
```

```
## [1] "sienaDependent"
```

Now that we have created our dependent variable (i.e. `advice` which is an object of class `sienaDependent`), we can create an object for RSiena to examine. We do this using the `sienaDataCreate` function. To see how this function works, look at the help with `?sienaDataCreate`. This may seem excessive *now*, but when we examine more networks simultaneously, the program architecture will make more sense.

```
# Take a look at the help.
?sienaDataCreate

# Now, build the object to analyze.
advice.data <- sienaDataCreate(advice)

advice.data
```

```
## Dependent variables:  advice
```

```
## Number of observations: 3
##
## Nodeset                    Actors
## Number of nodes                75
##
## Dependent variable advice
## Type               oneMode
## Observations       3
## Nodeset            Actors
## Densities          0.055 0.066 0.061
```

```
# advice.data is of class "siena".
class(advice.data)
```

```
## [1] "siena"
```

Step 2: Defining the set of effects to estimate using the `getEffects()` function

Now, we create an effects object for model specification using the `getEffects()` function. Basically, we are going to create an object with some effects, then as we continue to build our model, we can add or remove effects from that object. To see how this function works, look at the help with `?getEffects`.

By default, the `getEffects` function will estimate the rate of change in each period (i.e. the **rate** function). In this example, we will see two rates estimated, the rate of change from t1:t2 (rate 1) and t2:t3 (rate 2). Also, the model automatically adds the *outdegree* and *reciprocity* terms as these are necessary for estimation.

```
# Take a look at the help.
?getEffects

# Now, create the effects object.
advice.effects <- getEffects(advice.data)
```

Step 3: Create the model using the `sienaAlgorithmCreate` and `sienaModelCreate()` functions

Now that our **dependent** variable is defined (i.e. `advice.data`) and the effects we want to estimate on that object are defined (i.e. `advice.effects`), we can create a model object using the `sienaAlgorithmCreate()` or `sienaModelCreate()` functions. To see how these functions work, look at the help with `?sienaAlgorithmCreate` and `?sienaModelCreate`. These functions allow us to specify various properties of the estimation algorithm and the model.

```
# Take a look at the help.
?sienaAlgorithmCreate
?sienaModelCreate

# Create a model that has particulars about estimation, then we estimate the model.
advice.model <- sienaModelCreate(projname = "Advice", useStdInits=TRUE, seed=605)
advice.model
```

```
##  Siena Algorithm specification.
##  Project name: Advice
##  Use standard initial values: TRUE
##  Random seed: 605
##  Number of subphases in phase 2: 4
##  Starting value of gain parameter: 0.2
##  Reduction factor for gain parameter: 0.5
```

```
##  Diagonalization parameter: 0.2
##  Double averaging after subphase: 0
##  Dolby noise reduction: TRUE
##  Method for calculation of derivatives: Scores
##  Number of subphases in phase 2: 4
##  Number of iterations in phase 3: 1000
##  Unconditional simulation if more than one dependent variable
```

Taking Stock of the Steps (so far) and Checking Network Stability

We have gone through three steps in detail. But, note that we really only have to run several lines of command to get us to where we are right now. That is, ready to estimate a model. Here are all the lines we needed to get to the estimating stage:

```
advice         <- sienaDependent(array(c(advice1,advice2,advice3),dim=c(75,75,3)))
advice.data    <- sienaDataCreate(advice)
advice.effects <- getEffects(advice.data)
advice.model   <- sienaModelCreate(projname = "Advice", useStdInits=TRUE, seed=605)
```

In addition, we need to check whether there is sufficient stability in our network prior to estimating a model. If there is a lot of instability (i.e. there are dramatic changes between cross-sections), then it may provide difficult to model the evolution of the network. We can get a sense of how much change there is by examining the *Jaccard index*, for more information see the RSiena manual. A general rule of thumb is that the values should be above 0.3. The `print01Report()` function will return this information as a text file. Let's take a look:

```
?print01Report

# the report will have the file extension .out.
print01Report(advice.data, modelname = "Advice Example")
```

Step 4: Estimate the model using the `siena07` function

Now we are ready to estimate the model! To do this, we pass to the `siena07()` function the model information, the data, and the effects. To see how this function works, look at the help with `?siena07`. Now, let's estimate our model.

```
# Take a look at the help.
?siena07

# Estimate the model.
advice.results <- siena07(
    advice.model,           # the model estimation information.
    data=advice.data,       # the data object we created above.
    effects=advice.effects  # the effects object we created above.
)
```

```
## No X11 device available, forcing use of batch mode
## Start phase 0
## theta: -1.31  0.00
##
## Start phase 1
## Phase 1 Iteration 1 Progress: 0%
## Phase 1 Iteration 2 Progress: 0%
## Phase 1 Iteration 3 Progress: 0%
```

```
## Phase 1 Iteration 4 Progress: 0%
## Phase 1 Iteration 5 Progress: 0%
## Phase 1 Iteration 10 Progress: 0%
## Phase 1 Iteration 15 Progress: 1%
## Phase 1 Iteration 20 Progress: 1%
## Phase 1 Iteration 25 Progress: 1%
## Phase 1 Iteration 30 Progress: 1%
## Phase 1 Iteration 35 Progress: 1%
## Phase 1 Iteration 40 Progress: 2%
## Phase 1 Iteration 45 Progress: 2%
## Phase 1 Iteration 50 Progress: 2%
## theta: -1.340  0.142
##
## Start phase 2.1
## Phase 2 Subphase 1 Iteration 1 Progress: 6%
## Phase 2 Subphase 1 Iteration 2 Progress: 6%
## theta -1.365  0.299
## ac 0.784 1.135
## Phase 2 Subphase 1 Iteration 3 Progress: 6%
## Phase 2 Subphase 1 Iteration 4 Progress: 6%
## theta -1.454  0.789
## ac 1.04 1.20
## Phase 2 Subphase 1 Iteration 5 Progress: 6%
## Phase 2 Subphase 1 Iteration 6 Progress: 6%
## theta -1.53  1.19
## ac 1.15 1.21
## Phase 2 Subphase 1 Iteration 7 Progress: 6%
## Phase 2 Subphase 1 Iteration 8 Progress: 6%
## theta -1.59  1.41
## ac 1.22 1.17
## Phase 2 Subphase 1 Iteration 9 Progress: 6%
## Phase 2 Subphase 1 Iteration 10 Progress: 6%
## theta -1.62  1.48
## ac 1.20 1.09
## theta -1.59  1.30
## ac 0.166 0.832
## theta: -1.59  1.30
##
## Start phase 2.2
## Phase 2 Subphase 2 Iteration 1 Progress: 15%
## Phase 2 Subphase 2 Iteration 2 Progress: 15%
## Phase 2 Subphase 2 Iteration 3 Progress: 15%
## Phase 2 Subphase 2 Iteration 4 Progress: 15%
## Phase 2 Subphase 2 Iteration 5 Progress: 15%
## Phase 2 Subphase 2 Iteration 6 Progress: 15%
## Phase 2 Subphase 2 Iteration 7 Progress: 15%
## Phase 2 Subphase 2 Iteration 8 Progress: 15%
## Phase 2 Subphase 2 Iteration 9 Progress: 15%
## Phase 2 Subphase 2 Iteration 10 Progress: 15%
## theta -1.59  1.31
## ac -0.00379 -0.09686
## theta: -1.59  1.31
##
## Start phase 2.3
```

```
## Phase 2 Subphase 3 Iteration 1 Progress: 25%
## Phase 2 Subphase 3 Iteration 2 Progress: 25%
## Phase 2 Subphase 3 Iteration 3 Progress: 25%
## Phase 2 Subphase 3 Iteration 4 Progress: 25%
## Phase 2 Subphase 3 Iteration 5 Progress: 25%
## Phase 2 Subphase 3 Iteration 6 Progress: 25%
## Phase 2 Subphase 3 Iteration 7 Progress: 25%
## Phase 2 Subphase 3 Iteration 8 Progress: 25%
## Phase 2 Subphase 3 Iteration 9 Progress: 25%
## Phase 2 Subphase 3 Iteration 10 Progress: 25%
## theta -1.59  1.31
## ac 0.0883 0.0218
## theta: -1.59  1.31
##
## Start phase 2.4
## Phase 2 Subphase 4 Iteration 1 Progress: 38%
## Phase 2 Subphase 4 Iteration 2 Progress: 38%
## Phase 2 Subphase 4 Iteration 3 Progress: 39%
## Phase 2 Subphase 4 Iteration 4 Progress: 39%
## Phase 2 Subphase 4 Iteration 5 Progress: 39%
## Phase 2 Subphase 4 Iteration 6 Progress: 39%
## Phase 2 Subphase 4 Iteration 7 Progress: 39%
## Phase 2 Subphase 4 Iteration 8 Progress: 39%
## Phase 2 Subphase 4 Iteration 9 Progress: 39%
## Phase 2 Subphase 4 Iteration 10 Progress: 39%
## theta -1.59  1.30
## ac -0.000759 -0.016091
## theta: -1.59  1.30
##
## Start phase 3
## Phase 3 Iteration 500 Progress 80%
## Phase 3 Iteration 1000 Progress 100%
```

Interpreting the Output

Let's take a look through the results by using the `summary()` function.

```r
summary(advice.results)
```

```
## Estimates, standard errors and convergence t-ratios
##
##                                     Estimate   Standard   Convergence
##                                                  Error      t-ratio
##
## Rate parameters:
##   0.1      Rate parameter period 1  6.4262  ( 0.4847   )
##   0.2      Rate parameter period 2  5.2506  ( 0.3816   )
##
## Other parameters:
##   1.  eval outdegree (density)     -1.5909  ( 0.0416   )    0.0149
##   2.  eval reciprocity              1.3033  ( 0.1119   )   -0.0109
##
## Overall maximum convergence ratio:    0.0275
##
```

```
##
## Total of 2154 iteration steps.
##
## Covariance matrix of estimates (correlations below diagonal)
##
##        0.002       -0.002
##       -0.394        0.013
##
## Derivative matrix of expected statistics X by parameters:
##
##      562.325      202.220
##      103.578      161.442
##
## Covariance matrix of X (correlations below diagonal):
##
##      468.881      201.536
##        0.559      277.496
```

The output shows the estimates for the **rate** function and the estimates for the **objective** function. For each coefficient, we see an estimate and a standard error.

For the **objective** function coefficients, we also see the *Convergence t-ratio* reported. **NOTE**: these do not represent conventional *t-ratios* in the sence of a *t*-statistic assessing the size of the parameter estimate. Rather, they represent tests of the lack of convergence for each estimate, so small values indicate good convergence. Absolute values less than 0.10 indicate excellent convergence and absolute values less than 0.15 indicate reasonable convergence. More information about these values can be found in the RSiena manual.

The **rate** estimates correspond to the estimated number of opportunities for change per actor for each period. Recall that the model is simulating *micro-steps*, and in each micro-step an individual is provided the opportunity to make a decision. The **rate** parameter estimate gives a sense of how many opportunities are provided to each individual. For example, the estimate for period 1 is `6.4262`, meaning that each actor had just over 6 opportunities to change a tie or maintain their current tie configuration.

The "Other parameters" section shows the **objective** estimates, or the **eval** estimates, which correspond to the relative attractiveness of a particular network state for each actor. Recall that individuals are given a choice about what to do. These estimates represent the extent to which individuals preferred a particular state. For example, the `eval outdegree` term is negative, suggesting that individuals prefer *not* to send ties. The `eval reciprocity` term is positive, indicating that individuals prefer to reciprocate ties. Note that this preference is for *maintaining* an existing reciprocated relationship, for *creating* a reciprocated relationship from an asymmetric relationship where alter has nominated ego (i.e. ego wants to reciprocate), and for *dissolving* an asymmetric relationship where alter did not nominated ego after ego nominated alter (i.e. alter didn't reciprocate). The significance of the effects can be evaluated in a manner similar to a regression coefficient by looking at the ratio of the estimate to the standard error (where a ratio of 1.96 indicates a *p*-value of *0.05*).

Step 5: Adding Terms to the Model

When the effects object has been created, we can add to it using the `includeEffects()` function. This function allows us to take the existing effects object and add a specific term that we would like to estimate. To see how this function works, look at the help with `?includeEffects`. To see the effects, use the `effectsDocumentation()` function to generate an html file that shows all of the effects. Or, these terms can be found in the RSiena manual.

Let's add the following effects to our model:

- an effect for a preference for transitivity (i->j is preferred if i->k and k->j exists). This effect is called `transTrip`.
- an effect for a preference for 3-cycle triads (i->j is preferred if k->i and j->k exists). This effect is called `cycle3`.

```
# use the includeEffects() function to add these terms.
advice.effects <- includeEffects(advice.effects,transTrip) # the model term is "transTrip".
```

```
##   effectName          include fix   test  initialValue parm
## 1 transitive triplets TRUE    FALSE FALSE          0   0
```

```
advice.effects <- includeEffects(advice.effects,cycle3) # the model term is "cycle3"
```

```
##   effectName include fix   test  initialValue parm
## 1 3-cycles   TRUE    FALSE FALSE          0   0
```

Now that we have added new terms, we re-estimate the model.

```
advice.results2 <- siena07(advice.model,data=advice.data,effects=advice.effects)
```

```
## No X11 device available, forcing use of batch mode
## Start phase 0
## theta: -1.31  0.00  0.00  0.00
##
## Start phase 1
## Phase 1 Iteration 1 Progress: 0%
## Phase 1 Iteration 2 Progress: 0%
## Phase 1 Iteration 3 Progress: 0%
## Phase 1 Iteration 4 Progress: 0%
## Phase 1 Iteration 5 Progress: 0%
## Phase 1 Iteration 10 Progress: 0%
## Phase 1 Iteration 15 Progress: 1%
## Phase 1 Iteration 20 Progress: 1%
## Phase 1 Iteration 25 Progress: 1%
## Phase 1 Iteration 30 Progress: 1%
## Phase 1 Iteration 35 Progress: 1%
## Phase 1 Iteration 40 Progress: 1%
## Phase 1 Iteration 45 Progress: 2%
## Phase 1 Iteration 50 Progress: 2%
## theta: -1.4578  0.1178  0.0953  0.0318
##
## Start phase 2.1
## Phase 2 Subphase 1 Iteration 1 Progress: 9%
## Phase 2 Subphase 1 Iteration 2 Progress: 9%
## theta -1.5216  0.1964  0.1420  0.0482
## ac 0.816 0.940 1.015 1.082
## Phase 2 Subphase 1 Iteration 3 Progress: 9%
## Phase 2 Subphase 1 Iteration 4 Progress: 9%
## theta -1.7310  0.4752  0.2876  0.0747
## ac 0.716 1.009 0.876 0.406
## Phase 2 Subphase 1 Iteration 5 Progress: 9%
## Phase 2 Subphase 1 Iteration 6 Progress: 9%
## theta -1.8965  0.8624  0.3619 -0.0691
## ac 0.897 0.999 0.832 0.664
## Phase 2 Subphase 1 Iteration 7 Progress: 9%
## Phase 2 Subphase 1 Iteration 8 Progress: 9%
```

```
## theta -2.007  1.124  0.402 -0.182
## ac 0.864 0.982 0.773 0.605
## Phase 2 Subphase 1 Iteration 9 Progress: 9%
## Phase 2 Subphase 1 Iteration 10 Progress: 9%
## theta -2.078  1.268  0.409 -0.260
## ac 0.876 0.978 0.734 0.557
## theta -2.076  1.197  0.406 -0.222
## ac -0.0107  0.0438 -0.4060 -0.3689
## theta: -2.076  1.197  0.406 -0.222
##
## Start phase 2.2
## Phase 2 Subphase 2 Iteration 1 Progress: 17%
## Phase 2 Subphase 2 Iteration 2 Progress: 17%
## Phase 2 Subphase 2 Iteration 3 Progress: 17%
## Phase 2 Subphase 2 Iteration 4 Progress: 18%
## Phase 2 Subphase 2 Iteration 5 Progress: 18%
## Phase 2 Subphase 2 Iteration 6 Progress: 18%
## Phase 2 Subphase 2 Iteration 7 Progress: 18%
## Phase 2 Subphase 2 Iteration 8 Progress: 18%
## Phase 2 Subphase 2 Iteration 9 Progress: 18%
## Phase 2 Subphase 2 Iteration 10 Progress: 18%
## theta -2.078  1.193  0.410 -0.224
## ac -0.223 -0.267 -0.276 -0.228
## theta: -2.078  1.193  0.410 -0.224
##
## Start phase 2.3
## Phase 2 Subphase 3 Iteration 1 Progress: 27%
## Phase 2 Subphase 3 Iteration 2 Progress: 27%
## Phase 2 Subphase 3 Iteration 3 Progress: 27%
## Phase 2 Subphase 3 Iteration 4 Progress: 27%
## Phase 2 Subphase 3 Iteration 5 Progress: 27%
## Phase 2 Subphase 3 Iteration 6 Progress: 27%
## Phase 2 Subphase 3 Iteration 7 Progress: 27%
## Phase 2 Subphase 3 Iteration 8 Progress: 27%
## Phase 2 Subphase 3 Iteration 9 Progress: 27%
## Phase 2 Subphase 3 Iteration 10 Progress: 27%
## theta -2.080  1.196  0.411 -0.227
## ac -0.1804 -0.1691 -0.2100 -0.0689
## theta: -2.080  1.196  0.411 -0.227
##
## Start phase 2.4
## Phase 2 Subphase 4 Iteration 1 Progress: 41%
## Phase 2 Subphase 4 Iteration 2 Progress: 41%
## Phase 2 Subphase 4 Iteration 3 Progress: 41%
## Phase 2 Subphase 4 Iteration 4 Progress: 41%
## Phase 2 Subphase 4 Iteration 5 Progress: 41%
## Phase 2 Subphase 4 Iteration 6 Progress: 41%
## Phase 2 Subphase 4 Iteration 7 Progress: 41%
## Phase 2 Subphase 4 Iteration 8 Progress: 41%
## Phase 2 Subphase 4 Iteration 9 Progress: 41%
## Phase 2 Subphase 4 Iteration 10 Progress: 41%
## theta -2.080  1.198  0.408 -0.222
## ac -0.164 -0.218 -0.229 -0.143
## theta: -2.080  1.198  0.408 -0.222
```

```
##
## Start phase 3
## Phase 3 Iteration 500 Progress 82%
## Phase 3 Iteration 1000 Progress 100%
```

```r
summary(advice.results2)
```

```
## Estimates, standard errors and convergence t-ratios
##
##                                    Estimate   Standard   Convergence
##                                                 Error      t-ratio
##
## Rate parameters:
##   0.1      Rate parameter period 1  7.6743  ( 0.6680   )
##   0.2      Rate parameter period 2  5.6038  ( 0.4390   )
##
## Other parameters:
##   1.  eval outdegree (density)     -2.0804  ( 0.0577   )   -0.0076
##   2.  eval reciprocity              1.1980  ( 0.1253   )   -0.0125
##   3.  eval transitive triplets      0.4085  ( 0.0309   )   -0.0413
##   4.  eval 3-cycles                -0.2216  ( 0.0568   )   -0.0439
##
## Overall maximum convergence ratio:    0.0618
##
##
## Total of 1946 iteration steps.
##
## Covariance matrix of estimates (correlations below diagonal)
##
##        0.003       -0.002       -0.001        0.001
##       -0.288        0.016        0.000       -0.003
##       -0.636        0.096        0.001       -0.001
##        0.175       -0.437       -0.534        0.003
##
## Derivative matrix of expected statistics X by parameters:
##
##      871.886      333.384     2521.850      461.133
##      181.308      197.024      739.982      201.534
##     1749.590      839.838     8930.056     1717.178
##      523.200      400.669     3131.151      954.675
##
## Covariance matrix of X (correlations below diagonal):
##
##     1183.596      500.030     4410.024      824.660
##        0.713      415.643     2366.868      571.455
##        0.798        0.723    25803.360     5298.419
##        0.639        0.747        0.879     1407.568
```

What is the interpretation of the `transTrip` *and* `cycle3` *estimates?*

Let's add some additional effects to the model:

- An effect for seeking relationships with popular others (i->j is preferred if k->j exists. This effect is `inPop`.

- An effect for seeking relationships with active others (*i->j* is preferred if *j->k* exists). This effect is `outPop`.

```
# use the includeEffects() function to add these terms.
advice.effects <- includeEffects(advice.effects,inPop) # the model term is inPop (see also inPopSqrt)
```

```
##   effectName            include fix   test  initialValue parm
## 1 indegree - popularity TRUE    FALSE FALSE          0   0
```

```
advice.effects <- includeEffects(advice.effects,outPop) # the model term is outPop (see also outPopSqrt)
```

```
##   effectName             include fix   test  initialValue parm
## 1 outdegree - popularity TRUE    FALSE FALSE          0   0
```

Now that we have added new terms, we re-estimate the model.

```
advice.results3 <- siena07(advice.model,data=advice.data,effects=advice.effects)
```

```
## No X11 device available, forcing use of batch mode
## Start phase 0
## theta: -1.31  0.00  0.00  0.00  0.00  0.00
##
## Start phase 1
## Phase 1 Iteration 1 Progress: 0%
## Phase 1 Iteration 2 Progress: 0%
## Phase 1 Iteration 3 Progress: 0%
## Phase 1 Iteration 4 Progress: 0%
## Phase 1 Iteration 5 Progress: 0%
## Phase 1 Iteration 10 Progress: 0%
## Phase 1 Iteration 15 Progress: 1%
## Phase 1 Iteration 20 Progress: 1%
## Phase 1 Iteration 25 Progress: 1%
## Phase 1 Iteration 30 Progress: 1%
## Phase 1 Iteration 35 Progress: 1%
## Phase 1 Iteration 40 Progress: 1%
## Phase 1 Iteration 45 Progress: 2%
## Phase 1 Iteration 50 Progress: 2%
## theta: -1.44514  0.12707  0.08617  0.03064  0.00919 -0.01466
##
## Start phase 2.1
## Phase 2 Subphase 1 Iteration 1 Progress: 12%
## Phase 2 Subphase 1 Iteration 2 Progress: 12%
## theta -1.5078  0.2345  0.1409  0.0536  0.0125 -0.0237
## ac  0.607  1.091  1.098  1.095  1.160 -1.739
## Phase 2 Subphase 1 Iteration 3 Progress: 12%
## Phase 2 Subphase 1 Iteration 4 Progress: 12%
## theta -1.7038  0.6193  0.2984  0.0952  0.0250 -0.0577
## ac 0.87 1.14 1.11 1.05 1.14 1.05
## Phase 2 Subphase 1 Iteration 5 Progress: 12%
## Phase 2 Subphase 1 Iteration 6 Progress: 12%
## theta -1.84416  0.99932  0.36456  0.00947  0.03733 -0.08188
## ac 0.964 1.109 0.872 1.015 0.784 1.023
## Phase 2 Subphase 1 Iteration 7 Progress: 12%
## Phase 2 Subphase 1 Iteration 8 Progress: 12%
## theta -1.8817  1.2458  0.3305 -0.1402  0.0427 -0.0905
## ac 1.018 1.070 0.877 0.932 0.865 1.071
## Phase 2 Subphase 1 Iteration 9 Progress: 12%
```

```
## Phase 2 Subphase 1 Iteration 10 Progress: 12%
## theta -1.9547  1.4158  0.3687 -0.1045  0.0454 -0.1059
## ac 0.971 0.974 0.782 0.829 0.829 0.906
## theta -1.8870  1.3753  0.3533 -0.0409  0.0457 -0.1287
## ac  0.0186  0.0090 -0.2303 -0.3212 -0.2787 -0.1811
## theta: -1.8870  1.3753  0.3533 -0.0409  0.0457 -0.1287
##
## Start phase 2.2
## Phase 2 Subphase 2 Iteration 1 Progress: 20%
## Phase 2 Subphase 2 Iteration 2 Progress: 20%
## Phase 2 Subphase 2 Iteration 3 Progress: 20%
## Phase 2 Subphase 2 Iteration 4 Progress: 20%
## Phase 2 Subphase 2 Iteration 5 Progress: 20%
## Phase 2 Subphase 2 Iteration 6 Progress: 20%
## Phase 2 Subphase 2 Iteration 7 Progress: 20%
## Phase 2 Subphase 2 Iteration 8 Progress: 20%
## Phase 2 Subphase 2 Iteration 9 Progress: 20%
## Phase 2 Subphase 2 Iteration 10 Progress: 20%
## theta -1.9144  1.3653  0.3489 -0.0406  0.0454 -0.1196
## ac -0.149 -0.400 -0.516 -0.299 -0.294 -0.242
## theta: -1.9144  1.3653  0.3489 -0.0406  0.0454 -0.1196
##
## Start phase 2.3
## Phase 2 Subphase 3 Iteration 1 Progress: 29%
## Phase 2 Subphase 3 Iteration 2 Progress: 29%
## Phase 2 Subphase 3 Iteration 3 Progress: 29%
## Phase 2 Subphase 3 Iteration 4 Progress: 29%
## Phase 2 Subphase 3 Iteration 5 Progress: 29%
## Phase 2 Subphase 3 Iteration 6 Progress: 29%
## Phase 2 Subphase 3 Iteration 7 Progress: 29%
## Phase 2 Subphase 3 Iteration 8 Progress: 29%
## Phase 2 Subphase 3 Iteration 9 Progress: 29%
## Phase 2 Subphase 3 Iteration 10 Progress: 29%
## theta -1.8516  1.3783  0.3597 -0.0290  0.0466 -0.1402
## ac -0.0719 -0.1429 -0.2509 -0.1476 -0.1373 -0.1715
## theta: -1.8516  1.3783  0.3597 -0.0290  0.0466 -0.1402
##
## Start phase 2.4
## Phase 2 Subphase 4 Iteration 1 Progress: 43%
## Phase 2 Subphase 4 Iteration 2 Progress: 43%
## Phase 2 Subphase 4 Iteration 3 Progress: 43%
## Phase 2 Subphase 4 Iteration 4 Progress: 43%
## Phase 2 Subphase 4 Iteration 5 Progress: 43%
## Phase 2 Subphase 4 Iteration 6 Progress: 43%
## Phase 2 Subphase 4 Iteration 7 Progress: 43%
## Phase 2 Subphase 4 Iteration 8 Progress: 43%
## Phase 2 Subphase 4 Iteration 9 Progress: 43%
## Phase 2 Subphase 4 Iteration 10 Progress: 43%
## theta -1.8880  1.3791  0.3559 -0.0373  0.0463 -0.1305
## ac -0.0830 -0.0586 -0.0465 -0.0670 -0.0341 -0.0394
## theta: -1.8880  1.3791  0.3559 -0.0373  0.0463 -0.1305
##
## Start phase 3
## Phase 3 Iteration 500 Progress 83%
```

```
## Phase 3 Iteration 1000 Progress 100%
```

```r
summary(advice.results3)
```

```
## Estimates, standard errors and convergence t-ratios
##
##                                    Estimate   Standard   Convergence
##                                                 Error      t-ratio
##
## Rate parameters:
##   0.1      Rate parameter period 1  7.7780  ( 0.6876   )
##   0.2      Rate parameter period 2  5.8826  ( 0.4538   )
##
## Other parameters:
##   1.  eval outdegree (density)     -1.8880  ( 0.1359   )   0.0242
##   2.  eval reciprocity              1.3791  ( 0.1269   )   0.0541
##   3.  eval transitive triplets      0.3559  ( 0.0310   )   0.0319
##   4.  eval 3-cycles                -0.0373  ( 0.0657   )   0.0022
##   5.  eval indegree - popularity    0.0463  ( 0.0065   )   0.0648
##   6.  eval outdegree - popularity  -0.1305  ( 0.0411   )   0.0275
##
## Overall maximum convergence ratio:    0.1411
##
##
## Total of 2069 iteration steps.
##
## Covariance matrix of estimates (correlations below diagonal)
##
##        0.018        0.001        0.000        0.005        0.000       -0.005
##        0.062        0.016        0.001       -0.001        0.000       -0.001
##        0.097        0.152        0.001        0.000        0.000        0.000
##        0.577       -0.124       -0.083        0.004        0.000       -0.002
##        0.381        0.256       -0.024        0.440        0.000        0.000
##       -0.868       -0.255       -0.338       -0.633       -0.636        0.002
##
## Derivative matrix of expected statistics X by parameters:
##
##      696.465      281.855     2247.055      402.435    13859.326     7126.930
##      130.427      194.070      582.847      187.539     1374.001     1932.133
##     1521.678      739.412     8448.656     1593.341    36218.787    19038.253
##      351.942      383.376     2503.847      918.146     5141.743     6812.795
##     8237.045     2786.056    35214.086     5504.593   284161.518    94715.291
##     3657.628     1860.113    15345.134     3268.122    80425.686    45971.638
##
## Covariance matrix of X (correlations below diagonal):
##
##      885.015      383.240     3761.588      665.039    23509.871    10060.638
##        0.613      442.231     2193.200      622.965     7041.147     6173.650
##        0.816        0.673    24003.727     4979.148   109835.352    54334.477
##        0.583        0.773        0.838     1469.794    15275.965    12327.184
##        0.822        0.348        0.737        0.414   924689.934   259433.025
##        0.875        0.759        0.907        0.832        0.698   149506.936
```

What is the interpretation of the ***inPop*** *and* ***outPop*** *estimates?*

Working with Covariates

So far, we have specified terms that are primarily *structural* in the sense that they are concerned with endogenous network processes. However, we can also include covariates in the model to capture differential tie behavior based on the attributes of the actors.

In `RSiena`, covariates can be either:

- *Time-varying* in that the variable can vary across waves (e.g. smoking).
- *Constant* in that the variable does not change over time (e.g. sex).

Time-varying covariates are incorporated into the `sienaDataCreate()` function by defining them with the `varCovar()` function and constant covariates are incorporated using the `coCovar()` function.

Let's take a look at how we incorporate actor covariate terms by using the **performance ratings** of each student conducted at each wave. Since the performance ratings were different at each wave, it is a "time-varying covariate". So, we need to use the `varCovar()` function. Also, since we are only observing changes from *t1* to *t2* (i.e. period 1) and *t2* to *t3* (i.e. period 2), we only want the measurements taken at wave 1 and wave 2.

```
# call the data.
performance.data <- as.matrix(read.csv("https://www.jacobtnyoung.com/uploads/2/3/4/5/23459640/mba-perfo

# See the description of the varCovar() function.
?varCovar

# We have only take the wave 1 and 2 data, not wave 3 (because we don't observe changes during wave 3).
performance <- varCovar(performance.data[,1:2])
```

Now that we have the performance data, let's rebuild the objects we want. Since the `performance` object is data used in our analysis, we need to redefine the object created by the `sienaDataCreate()` function. Also, since we will be creating a new data object for analysis, we need to redefine the effects we want to estimate using the `getEffects()` function.

```
# add the performance data and create a new object.
a.p.data <- sienaDataCreate(advice, performance)
a.p.data
```

```
## Dependent variables:  advice
## Number of observations: 3
##
## Nodeset                  Actors
## Number of nodes              75
##
## Dependent variable advice
## Type               oneMode
## Observations       3
## Nodeset            Actors
## Densities          0.055 0.066 0.061
##
## Changing covariates:  performance
```

```
# We also need a new effects object.
a.p.effects <- getEffects(a.p.data)

# Let's use the terms we already had in the prior model.
# Note: we can just include all the effects in one statement, not four lines.
a.p.effects <- includeEffects(a.p.effects,transTrip,cycle3,inPop,outPop)
```

```
##   effectName              include fix   test  initialValue parm
## 1 transitive triplets     TRUE    FALSE FALSE          0   0
## 2 3-cycles                TRUE    FALSE FALSE          0   0
## 3 indegree - popularity   TRUE    FALSE FALSE          0   0
## 4 outdegree - popularity  TRUE    FALSE FALSE          0   0
```

Note that the `includeEffects()` function includes the argument `include=` which can be used to remove terms. For example, say we estimate a model and want to remove the `cycle3` effect. We would use the following command: `includeEffects(a.p.effects,cycle3, include=FALSE)`. If we did that and wanted to add it back in, we could use: `includeEffects(a.p.effects,cycle3, include=TRUE)` (note that the `include=TRUE` argument is not necessary as the default is `TRUE` for this argument).

Now we want to include the effects of `performance` on tie behavior. For actor covariates, we call these *interaction terms* because the outdegree depends on a covariate.

The typical effects we might be interested in are:

- Individuals with a particular attribute are more likely to *send* ties, a *sender* effect. This effect is called `egoX`.
- Individuals with a particular attribute are more likely to *receive* ties, a *receiver* effect. This effect is called `alterX`.
- Individuals with a particular attribute are more likely to nominate others with the same or similar attribute, a *homophily* effect. This effect is called `sameX` or `simX`.

Let's take a look at how this looks in the syntax and what the effects mean.

```r
# Hypothesis: low performers need advice.
  # So, people who have higher performance are less likely to send ties.
  # Should see a negative coefficient.
a.p.effects <- includeEffects(a.p.effects,
                              egoX, # sender effect.
                              interaction1="performance" # outdegree depends on performance.
                              )
```

```
##   effectName      include fix   test  initialValue parm
## 1 performance ego TRUE    FALSE FALSE          0   0
```

```r
# Hypothesis: high performers are asked for advice.
  # So, people who have higher performance are more likely to receive ties.
  # Should see a positive coefficient.
a.p.effects <- includeEffects(a.p.effects,
                              altX, # receiver effect.
                              interaction1="performance" # indegree depends on performance.
                              )
```

```
##   effectName        include fix   test  initialValue parm
## 1 performance alter TRUE    FALSE FALSE          0   0
```

```r
# Hypotheses: people at the same performance level do not seek advice from each other.
  # This is a homophily effect.
  # Since the variable is continuous, we should anticipate a positive effect to indicate homophily.
    # NOTE: because the performance measure is not categorical, we use "simX", not "sameX".
    # If our variable was categorical, we would use "sameX" and expect a positive coefficient.
a.p.effects <- includeEffects(a.p.effects,simX,interaction1="performance")
```

```
##   effectName              include fix   test  initialValue parm
## 1 performance similarity  TRUE    FALSE FALSE          0   0
```

Now that we have our effects object described, let's estimate the model.

```
# Set up the estimation.
a.p.model <- sienaModelCreate(projname = "Advice & Performance", useStdInits=TRUE, seed=605)

# Estimate the model.
advice.results <- siena07(a.p.model,data=a.p.data,effects= a.p.effects)
```

```
## No X11 device available, forcing use of batch mode
## Start phase 0
## theta: -1.31  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
##
## Start phase 1
## Phase 1 Iteration 1 Progress: 0%
## Phase 1 Iteration 2 Progress: 0%
## Phase 1 Iteration 3 Progress: 0%
## Phase 1 Iteration 4 Progress: 0%
## Phase 1 Iteration 5 Progress: 0%
## Phase 1 Iteration 10 Progress: 0%
## Phase 1 Iteration 15 Progress: 0%
## Phase 1 Iteration 20 Progress: 1%
## Phase 1 Iteration 25 Progress: 1%
## Phase 1 Iteration 30 Progress: 1%
## Phase 1 Iteration 35 Progress: 1%
## Phase 1 Iteration 40 Progress: 1%
## Phase 1 Iteration 45 Progress: 1%
## Phase 1 Iteration 50 Progress: 2%
## theta: -1.45832  0.12897  0.08647  0.03259  0.00908 -0.01199  0.00120 -0.00647  0.00261
##
## Start phase 2.1
## Phase 2 Subphase 1 Iteration 1 Progress: 15%
## Phase 2 Subphase 1 Iteration 2 Progress: 15%
## theta -1.5290  0.2501  0.1498  0.0597  0.0118 -0.0213  0.0129 -0.0110  0.0323
## ac    0.600    1.018    1.187    1.292    1.158 -29.294    1.145    0.539    0.955
## Phase 2 Subphase 1 Iteration 3 Progress: 15%
## Phase 2 Subphase 1 Iteration 4 Progress: 15%
## theta -1.7411  0.7003  0.2938  0.0625  0.0238 -0.0509  0.0380 -0.0276  0.2019
## ac 0.972 1.047 1.148 1.183 0.878 1.167 1.146 0.626 0.977
## Phase 2 Subphase 1 Iteration 5 Progress: 15%
## Phase 2 Subphase 1 Iteration 6 Progress: 15%
## theta -1.8811  1.1100  0.3255 -0.0469  0.0338 -0.0711  0.0418 -0.0522  0.4693
## ac 1.061 1.056 1.190 1.282 0.886 1.279 1.158 1.042 0.898
## Phase 2 Subphase 1 Iteration 7 Progress: 15%
## Phase 2 Subphase 1 Iteration 8 Progress: 15%
## theta -1.9752  1.3840  0.3323 -0.1094  0.0382 -0.0831  0.0461 -0.0759  0.7519
## ac 1.071 1.051 1.182 1.283 0.898 1.275 1.127 1.041 0.912
## Phase 2 Subphase 1 Iteration 9 Progress: 15%
## Phase 2 Subphase 1 Iteration 10 Progress: 15%
## theta -2.0488  1.5745  0.3546 -0.1557  0.0445 -0.0970  0.0539 -0.0944  0.9108
## ac 1.074 1.053 1.181 1.183 0.869 1.255 1.115 0.765 0.908
## theta -1.9750  1.4031  0.3354 -0.0361  0.0408 -0.1105  0.0758 -0.1058  1.0813
## ac -0.215 -0.186 -0.485 -0.386 -0.450 -0.332 -0.259 -0.421  0.143
## theta: -1.9750  1.4031  0.3354 -0.0361  0.0408 -0.1105  0.0758 -0.1058  1.0813
##
## Start phase 2.2
```

```
## Phase 2 Subphase 2 Iteration 1 Progress: 22%
## Phase 2 Subphase 2 Iteration 2 Progress: 22%
## Phase 2 Subphase 2 Iteration 3 Progress: 22%
## Phase 2 Subphase 2 Iteration 4 Progress: 22%
## Phase 2 Subphase 2 Iteration 5 Progress: 22%
## Phase 2 Subphase 2 Iteration 6 Progress: 22%
## Phase 2 Subphase 2 Iteration 7 Progress: 22%
## Phase 2 Subphase 2 Iteration 8 Progress: 22%
## Phase 2 Subphase 2 Iteration 9 Progress: 23%
## Phase 2 Subphase 2 Iteration 10 Progress: 23%
## theta -1.9866  1.3918  0.3360 -0.0392  0.0409 -0.1072  0.0723 -0.1042  1.0943
## ac -0.26312 -0.30113 -0.29519 -0.25120 -0.30333 -0.18139 -0.05286 -0.01547 -0.00542
## theta: -1.9866  1.3918  0.3360 -0.0392  0.0409 -0.1072  0.0723 -0.1042  1.0943
##
## Start phase 2.3
## Phase 2 Subphase 3 Iteration 1 Progress: 31%
## Phase 2 Subphase 3 Iteration 2 Progress: 31%
## Phase 2 Subphase 3 Iteration 3 Progress: 31%
## Phase 2 Subphase 3 Iteration 4 Progress: 31%
## Phase 2 Subphase 3 Iteration 5 Progress: 31%
## Phase 2 Subphase 3 Iteration 6 Progress: 31%
## Phase 2 Subphase 3 Iteration 7 Progress: 31%
## Phase 2 Subphase 3 Iteration 8 Progress: 31%
## Phase 2 Subphase 3 Iteration 9 Progress: 32%
## Phase 2 Subphase 3 Iteration 10 Progress: 32%
## theta -1.9663  1.3903  0.3336 -0.0313  0.0413 -0.1110  0.0734 -0.1046  1.0799
## ac -0.0578 -0.1816 -0.1138 -0.1746 -0.0432 -0.1268 -0.1250  0.0627  0.0104
## theta: -1.9663  1.3903  0.3336 -0.0313  0.0413 -0.1110  0.0734 -0.1046  1.0799
##
## Start phase 2.4
## Phase 2 Subphase 4 Iteration 1 Progress: 45%
## Phase 2 Subphase 4 Iteration 2 Progress: 45%
## Phase 2 Subphase 4 Iteration 3 Progress: 45%
## Phase 2 Subphase 4 Iteration 4 Progress: 45%
## Phase 2 Subphase 4 Iteration 5 Progress: 45%
## Phase 2 Subphase 4 Iteration 6 Progress: 45%
## Phase 2 Subphase 4 Iteration 7 Progress: 45%
## Phase 2 Subphase 4 Iteration 8 Progress: 45%
## Phase 2 Subphase 4 Iteration 9 Progress: 45%
## Phase 2 Subphase 4 Iteration 10 Progress: 45%
## theta -1.9798  1.3939  0.3364 -0.0333  0.0411 -0.1091  0.0727 -0.1046  1.0741
## ac -0.02402 -0.02217 -0.00960 -0.03493 -0.05688  0.01717 -0.10621  0.00865  0.07683
## theta: -1.9798  1.3939  0.3364 -0.0333  0.0411 -0.1091  0.0727 -0.1046  1.0741
##
## Start phase 3
## Phase 3 Iteration 500 Progress 85%
## Phase 3 Iteration 1000 Progress 100%
```

```r
# Take a look at the output.
summary(advice.results)
```

```
## Estimates, standard errors and convergence t-ratios
##
##                                     Estimate   Standard   Convergence
##                                                  Error      t-ratio
```

```
##
## Rate parameters:
##   0.1      Rate parameter period 1  8.0042  ( 0.7076   )
##   0.2      Rate parameter period 2  5.9324  ( 0.4610   )
##
## Other parameters:
##   1.  eval outdegree (density)     -1.9798  ( 0.1210   )   0.0714
##   2.  eval reciprocity              1.3939  ( 0.1355   )   0.0431
##   3.  eval transitive triplets      0.3364  ( 0.0289   )   0.1003
##   4.  eval 3-cycles                -0.0333  ( 0.0617   )   0.0724
##   5.  eval indegree - popularity    0.0411  ( 0.0067   )   0.0677
##   6.  eval outdegree - popularity  -0.1091  ( 0.0335   )   0.0643
##   7.  eval performance alter        0.0727  ( 0.0270   )   0.0422
##   8.  eval performance ego         -0.1046  ( 0.0277   )   0.0056
##   9.  eval performance similarity   1.0741  ( 0.3079   )   0.0233
##
## Overall maximum convergence ratio:    0.1277
##
##
## Total of 2683 iteration steps.
##
## Covariance matrix of estimates (correlations below diagonal)
##
##       0.015        0.001        0.000        0.003        0.000       -0.003        0.000        0.0
##       0.089        0.018        0.000       -0.002        0.000       -0.001        0.000       -0.0
##      -0.080        0.078        0.001       -0.001        0.000        0.000        0.000        0.0
##       0.373       -0.245       -0.305        0.004        0.000       -0.001        0.000        0.0
##       0.214        0.239       -0.148        0.310        0.000        0.000        0.000        0.0
##      -0.807       -0.298       -0.163       -0.452       -0.551        0.001        0.000        0.0
##      -0.130        0.081       -0.061       -0.042       -0.331        0.115        0.001        0.0
##      -0.001       -0.185       -0.011       -0.258       -0.171        0.204       -0.207        0.0
##      -0.055        0.084        0.039        0.076        0.150       -0.115        0.117       -0.5
##
## Derivative matrix of expected statistics X by parameters:
##
##     724.618      286.174     2346.153      446.190    14128.239     7425.386      621.793       49.7
##     144.156      201.454      732.359      235.414     1769.901     2298.373       67.210      124.7
##    1591.376      808.407     8997.500     1761.441    36468.703    20148.535     1742.183      483.8
##     423.226      437.371     2955.157     1030.360     6570.750     7832.623      319.431      429.5
##    8094.651     2717.282    33693.110     5185.846   263008.645    89340.170    11032.980     1100.3
##    3720.704     1851.336    15152.051     3271.746    78337.218    46218.225     3394.665      582.7
##     643.007      187.490     2942.657      436.325    19801.239     7331.355     2200.800      566.5
##      12.848      224.023     1151.803      452.373       96.974     3272.721      269.131     2051.0
##      14.107       14.982       63.387       22.779        7.170      228.310       -1.332       87.8
##
## Covariance matrix of X (correlations below diagonal):
##
##     899.363      402.254     3668.799      690.782    21633.574    10156.631      952.575      154.9
##       0.642      436.082     2276.882      653.727     7203.893     6406.886      324.357      315.0
##       0.793        0.707    23797.383     5136.165    97992.536    53855.246     4732.717     1946.4
##       0.584        0.794        0.844     1554.445    13902.227    12722.886      689.908      636.1
##       0.816        0.390        0.719        0.399   781563.523   233311.649    31363.012     2374.1
##       0.874        0.792        0.901        0.833        0.681   150181.842    11153.489     5246.7
##       0.629        0.307        0.607        0.346        0.702        0.570     2551.817      534.4
```

```
##          0.116          0.339          0.283          0.362          0.060          0.304          0.238      1984.
##          0.155          0.292          0.184          0.255          0.013          0.230          0.010          0.
```

Now let's interpret the effects:

- For `eval performance ego` we see a negative coefficient, indicating that individuals with higher values on their performance rating were **less** likely to *send* ties.
- For `eval performance alter` we see a positive coefficient, indicating that individuals with higher values on their performance rating were **more** likely to *receive* ties.
- For `eval performance similarity` we see a positive coefficient, indicating that ego prefers sending ties to alters who are more similar to ego on the rating variable.

Questions?