STERGM - Separable Temporal ERGMs for modeling discrete relational dynamics with *statnet*

Pavel N. Krivitsky, Steven M. Goodreau,
The `Statnet` Development Team

May 17, 2013

Contents

1 Getting the software

If you have not already done so, please make sure that you have a reasonably new version of R, preferably the latest (3.0.0). Then, download and install the latest versions of the statnet packages. You will specifically need to have version 3.1-0 of the packages `ergm`, `tergm` and `statnet.common`, as well as `network` version 1.7.2 and `networkDynamic` version 0.4 or later. You can get all of them with the syntax:

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

2 A quick review of static ERGMs

Exponential-family random graph models (ERGMs) represent a general class of models based in exponential-family theory for specifying the probability distribution underlying a set of random graphs or networks. Within this framework, one can—among other tasks—obtain maximum-likehood estimates for the parameters of a specified model for a given data set; simulate additional networks with the underlying probability distribution implied by that model; test individual models for goodness-of-fit, and perform various types of model comparison.

The basic expression for the ERGM class can be written as:

$$P(Y = y) = \frac{\exp(\theta' g(y))}{k(y)} \tag{1}$$

where Y is the random variable for the state of the network (with realization y), $g(y)$ is the vector of model statistics for network y, θ is the vector of coefficients for those statistics, and $k(y)$ represents the quantity in the numerator summed over all possible networks (typically constrained to be all networks with the same node set as y).

This can be re-expressed in terms of the conditional log-odds of a single actor pair:

$$\text{logit}\,(Y_{ij} = 1 | y^c_{ij}) = \theta' \delta(y_{ij}) \tag{2}$$

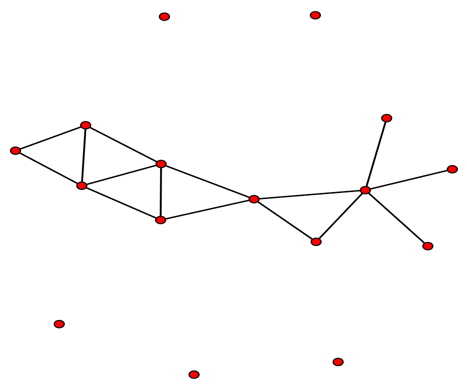
where Y_{ij} is the random variable for the state of the actor pair i, j (with realization y_{ij}), and y^c_{ij} signifies the complement of y_{ij}, i.e. all dyads in the network other than y_{ij}. The variable $\delta(y_{ij})$ equals $g(y_{ij}^+) - g(y_{ij}^-)$, where y_{ij}^+ is defined as y_{ij}^c along with y_{ij} set to 1, and y_{ij}^- is defined as y_{ij}^c along with y_{ij} set to 0. That is, $\delta(y_{ij})$ equals the value of $g(y)$ when $y_{ij} = 1$ minus the value of $g(y)$ when $y_{ij} = 0$, but all other dyads are as in $g(y)$. This emphasizes the log-odds of an individual tie conditional on all others. We call $g(y)$ the statistics of the model, and $\delta(y_{ij})$ the "change statistics" for actor pair y_{ij}.

Fitting an ERGM usually begins with obtaining data:

```
> library(tergm)
> data("florentine")
> ls()
```

```
[1] "flobusiness" "flomarriage"
```

```
> plot(flobusiness)
```



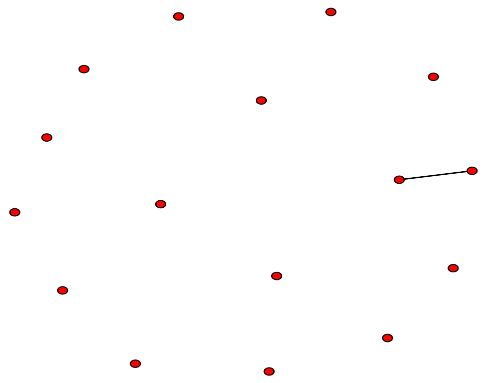
To refresh our memories on ERGM syntax, let us fit a cross-sectional example. Just by looking at the plot of `flobusiness`, we might guess that there are more triangles than expected by chance for a network of this size and density, and thus that there is some sort of explicit triangle closure effect going on. One useful way to model this effect in ERGMs that has been explored in the literature is with a `gwesp` statistic.

```
> fit1 <- ergm(flobusiness~edges+gwesp(0,fixed=T))

Iteration 1 of at most 20:
Convergence test P-value: 1.6e-253
The log-likelihood improved by 0.07054
Iteration 2 of at most 20:
Convergence test P-value: 5e-45
The log-likelihood improved by 0.01656
Iteration 3 of at most 20:
Convergence test P-value: 3.9e-06
The log-likelihood improved by 0.002125
Iteration 4 of at most 20:
Convergence test P-value: 3.5e-04
The log-likelihood improved by 0.001339
Iteration 5 of at most 20:
Convergence test P-value: 7.1e-01
Convergence detected. Stopping.
The log-likelihood improved by < 0.0001

This model was fit using MCMC.  To examine model diagnostics and check for degeneracy

> summary(fit1)

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

Formula:   flobusiness ~ edges + gwesp(0, fixed = T)

Iterations:  20

Monte Carlo MLE Results:
              Estimate Std. Error MCMC % p-value
edges          -3.3674     0.6128      0 < 1e-04 ***
gwesp.fixed.0   1.5646     0.5810      0 0.00812 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 166.36  on 120  degrees of freedom
 Residual Deviance:  78.21  on 118  degrees of freedom
```

```
AIC: 82.21    BIC: 87.79    (Smaller is better.)
```

With the estimation in place, we can simulate a new network from the given model:

```
> sim1 <- simulate(fit1,nsim=1,
           control=control.simulate.ergm(MCMC.burnin=1000))
> plot(sim1)
```



3 An Introduction to STERGMs (non-technical)

Separable Temporal ERGMs (STERGMs) are an extension of ERGMs for modeling dynamic networks in discrete time, introduced in Krivitsky and Handcock (2010). The cross-sectional ERGM entails a single network, and a single model on that network. STERGMs, in contrast, posit two models: one ERGM underlying relational formation, and a second one underlying relational dissolution. Specifying a STERGM thus entails writing two ERGM formulas instead of one. It also requires dynamic data, of course; such data can come in many forms, and we will cover a few examples today.

This approach is not simply a methodological development, but a theoretical one as well, and one which matches common sense for many social processes. Think of romantic relations. It seems intuitive that the statistical model underlying relational formation (i.e. affecting who becomes partners with whom, out of the set of people who aren't already) is likely to be different than the model underlying relational dissolution (i.e. affecting who breaks up with whom, out of the set of people currently in relationships). Any reasonable model of the former would need to include a variety of homophily parameters (mixing on age, for example). The latter may or may not. (Conditional on being in a relationship, does your difference in age affect your probability of breaking up? Perhaps, but probably not as fundametally or as strongly as for formation).

4 An Introduction to STERGMs (a bit more technical)

We first review the ERGM framework for *cross-sectional* or static networks, observed at a single point in time. Let $\mathbb{Y} \subseteq \{1, \dots, n\}^2$ be the set of potential relations (dyads) among n nodes, ordered for directed networks and unordered for undirected. We can represent a network $\mathbf{y}$ as a set of ties, with the set of possible sets of ties, $\mathcal{Y} \subseteq 2^{\mathbb{Y}}$, being the sample space: $\mathbf{y} \in \mathcal{Y}$. Let $\mathbf{y}_{ij}$ be 1 if $(i, j) \in \mathbf{y}$ — a relation of interest exists from i to j — and 0 otherwise.

The network also has an associated covariate array $\mathbf{X}$ containing attributes of the nodes, the dyads, or both. An exponential-family random graph model (ERGM) represents the pmf of $\mathbf{Y}$ as a function of a p-vector of network statistics $g(\mathbf{Y}, \mathbf{X})$, with parameters $\theta \in \mathbb{R}^p$, as follows:

$$\Pr(\mathbf{Y} = \mathbf{y} \mid \mathbf{X}) = \frac{\exp\{\theta \cdot g(\mathbf{y}, \mathbf{X})\}}{k(\theta, \mathbf{X}, \mathcal{Y})}, \tag{3}$$

where the normalizing constant

$$k(\theta, \mathbf{X}, \mathcal{Y}) = \sum_{\mathbf{y}' \in \mathcal{Y}} \exp\left\{\theta \cdot g(\mathbf{y}', \mathbf{X})\right\}$$

is a summation over the space of possible networks on n nodes, $\mathcal{Y}$. Where $\mathcal{Y}$ and $\mathbf{X}$ are held constant, as in a typical cross-sectional model, they may be suppressed in the notation. Here, on the other hand, the dependence on $\mathcal{Y}$ and $\mathbf{X}$ is made explicit.

In modeling the transition from a network $\mathbf{Y}^t$ at time t to a network $\mathbf{Y}^{t+1}$ at time $t+1$, the separable temporal ERGM assumes that the formation and dissolution of ties occur independently from each other within each time step, with each half of the process modeled as an ERGM. For two networks (sets of ties) $\mathbf{y}, \mathbf{y}' \in \mathcal{Y}$, let $\mathbf{y} \supseteq \mathbf{y}'$ if any tie present in $\mathbf{y}'$ is also present in $\mathbf{y}$. Define $\mathcal{Y}^+(\mathbf{y}) = \{\mathbf{y}' \in \mathcal{Y} : \mathbf{y}' \supseteq \mathbf{y}\}$, the networks that can be constructed by forming ties in $\mathbf{y}$; and $\mathcal{Y}^-(\mathbf{y}) = \{\mathbf{y}' \in \mathcal{Y} : \mathbf{y}' \subseteq \mathbf{y}\}$, the networks that can be constructed dissolving ties in $\mathbf{y}$.

Given $\mathbf{y}^t$, a *formation network* $\mathbf{Y}^+$ is generated from an ERGM controlled by a p-vector of formation parameters θ^+ and formation statistics $g^+(\mathbf{y}^+, \mathbf{X})$, conditional on only adding ties:

$$\Pr\left(\mathbf{Y}^+ = \mathbf{y}^+ \mid \mathbf{Y}^t; \theta^+\right) = \frac{\exp\left\{\theta^+ \cdot g^+(\mathbf{y}^+, \mathbf{X})\right\}}{k\left(\theta^+, \mathbf{X}, \mathcal{Y}^+(\mathbf{Y}^t)\right)}, \quad \mathbf{y}^+ \in \mathcal{Y}^+(\mathbf{y}^t). \tag{4}$$

A *dissolution network* $\mathbf{Y}^-$ is simultaneously generated from an ERGM controlled by a (possibly different) q-vector of dissolution parameters θ^- and corresponding statistics $g^-(\mathbf{y}^-, \mathbf{X})$, conditional on only dissolving ties from $\mathbf{y}^t$:

$$\Pr\left(\mathbf{Y}^- = \mathbf{y}^- \mid \mathbf{Y}^t; \theta^-\right) = \frac{\exp\left\{\theta^- \cdot g^-(\mathbf{y}^-, \mathbf{X})\right\}}{k\left(\theta^-, \mathbf{X}, \mathcal{Y}^-(\mathbf{Y}^t)\right)}, \quad \mathbf{y}^- \in \mathcal{Y}^-(\mathbf{y}^t). \tag{5}$$

The cross-sectional network at time $t+1$ is then constructed by applying the changes in $\mathbf{Y}^+$ and $\mathbf{Y}^-$ to $\mathbf{y}^t$:

$$\mathbf{Y}^{t+1} = \mathbf{Y}^t \cup (\mathbf{Y}^+ - \mathbf{Y}^t) - (\mathbf{Y}^t - \mathbf{Y}^-).$$

which simplifies to either:

$$\mathbf{Y}^{t+1} = \mathbf{Y}^+ - (\mathbf{Y}^t - \mathbf{Y}^-)$$

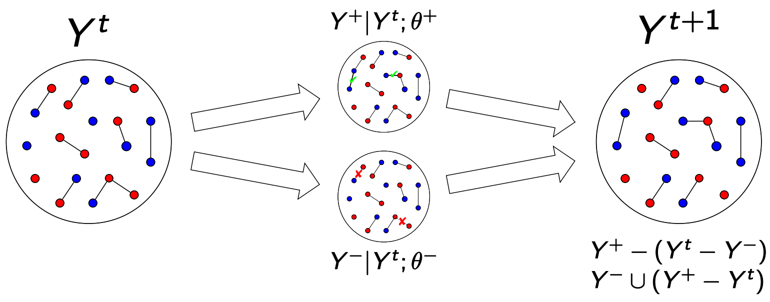
$$\mathbf{Y}^{t+1} = \mathbf{Y}^- \cup (\mathbf{Y}^+ - \mathbf{Y}^t)$$

Visually, we can sum this up as:

5 Notes on model specification and syntax

Within *statnet*, an ERGM involves one network and one set of network statistics, so these are specified together using R's formula notation:

```
my.network ~ my.vector.of.model.terms
```



For a call to stergm, there is still one network, but two formulas. These are now passed as three separate arguments: the network (argument nw), the formation formula (argument formation), and the dissolution formula (argument dissolution). The latter two both take the form of a one-sided formula. E.g.:

```
stergm(my.network,
    formation= ~edges+kstar(2),
    dissolution= ~edges+triangle
)
```

There are other features of a call to either ergm or stergm that will be important for us here. We list the features here; each will be illustrated in one or more examples during the workshop.

1. To fix the coefficient for a particular network statistic, one uses offset notation. For instance, to fix a dissolution model with only an edges term with parameter value 4.2, the dissolution formula woud be:

   ```
   dissolution= ∼offset(edges)
   ```

   and the corresponding argument for passing the parameter value would be:

   ```
   offset.coef.diss = 4.2
   ```

2. In parallel with ergm, any information used to specify the nature of the fitting algorithm is passed by specifying a vector called control.stergm to the control argument. For example:

```
control=control.stergm(MCMC.burnin=10000)
```

For a list of options, type `?control.stergm`

3. Another argument that the user must supply is `estimate`, which controls the estimation method. Unlike with cross-sectional ERGMs, there is not necessarily an obvious default here, as different scenarios are best fit with different approaches. The most important for the new user to recognize are `EGMME` (equilibrium generalized method of moments estimation) and `CMLE` (conditional maximum likelihood estimation). A good rule of thumb is that when fitting to two (or more) networks (i.e. with panel data), one should use `estimate="CMLE"`; and when fitting to a single cross-section with some duration information, use `estimate="EGMME"`.

6 Example 1: Multiple network cross-sections

One common form of data for modeling dynamic network processes consists of observations of network structure at two or more points in time on the same node set. Many classic network studies were of this type, and data of this form continue to be collected and analyzed.

Let us consider the famous Sampson monastery data:

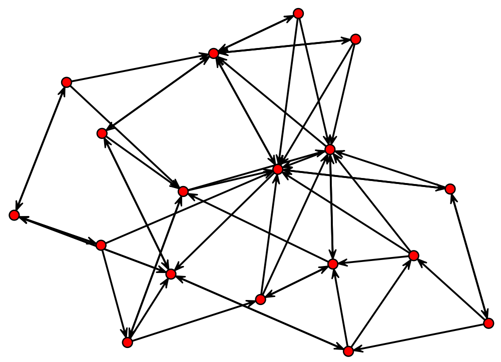
```
> data(samplk)
> ls()

[1] "fit1"        "flobusiness" "flomarriage" "samplk1"
[5] "samplk2"     "samplk3"     "sim1"
```

To pass them into `stergm`, we need to combine them into a list:

```
> samp <- list()
> samp[[1]] <- samplk1
> samp[[2]] <- samplk2
> samp[[3]] <- samplk3
```

Now we must decide on a model to fit to them. Plotting one network:

```
> plot(samplk1)
```



we might get the idea to consider mutuality as a predictor of a directed edge. Also, since this is a directed network, and there appear to be a considerable number of triadic relations, it might be worth investigating the role of cyclicity vs. transitivity in the network. Although there are a number of ways to model this latter question, we will select the relatively robust measures “transitiveties” and “cyclicalties” (the number of ties i->j that have at least one two-path from i->j and from j->i, respectively). Since we have multiple network snapshots, and we have separate formation and dissolution models, we can estimate the degree to which closing a mutual dyad or closing a triangle of each type predicts the creation of a tie, and also estimate the degree to which maintaining a mutual dyad or maintaining a triad of each type predicts the persistence of an existing tie. We might see different phenomena at work in each case; or the same phenomena, but with different coefficients.

In the following code, we pass five arguments: the series of networks that we want to model (`samp`), the formation formula, the dissolution formula, the fitting algorithm (in this case, conditional MLE is best since we have a network time series), and the list of time slices we wish to model (which includes all in the passed network series by default):

```
> samp.fit <- stergm(samp,
          formation= ~edges+mutual+cyclicalties+transitiveties,
          dissolution = ~edges+mutual+cyclicalties+transitiveties,
          estimate = "CMLE",
   times=1:3
          )

   Fitting formation:
Iteration 1 of at most 20:
```

⇒ Lots of output snipped. ⇐

```
This model was fit using MCMC. To examine model diagnostics and
check for degeneracy, use the mcmc.diagnostics() function.
```

And the results:

```
> summary(samp.fit)

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

Formula:   ~edges + mutual + cyclicalties + transitiveties

Iterations:  20

Monte Carlo MLE Results:
               Estimate Std. Error MCMC % p-value
edges           -3.4696     0.3351      0  <1e-04 ***
mutual           2.0464     0.4105      0  <1e-04 ***
cyclicalties    -0.1365     0.2013      0  0.4981
transitiveties   0.3927     0.2357      0  0.0963 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 693.1  on 500  degrees of freedom
 Residual Deviance: 239.9  on 496  degrees of freedom

AIC: 247.9    BIC: 264.8    (Smaller is better.)
```

```
================================
Summary of dissolution model fit
================================

Formula:   ~edges + mutual + cyclicalties + transitiveties

Iterations:  20

Monte Carlo MLE Results:
               Estimate Std. Error MCMC % p-value
edges            0.2083     0.3022      0  0.4920
mutual           0.7955     0.5166      0  0.1265
cyclicalties    -0.1893     0.2502      0  0.4509
transitiveties   0.5028     0.2834      0  0.0788 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 155.3  on 112  degrees of freedom
 Residual Deviance: 136.6  on 108  degrees of freedom

AIC: 144.6    BIC: 155.5    (Smaller is better.)
```

So, all else being equal, a relationship is much more likely to form if it will close a mutual pair. The effect on persistence is also positive, but less strong and clear. Transitivity may also matter, perhaps slightly more so in terms of dissolution than formation.

If one wishes to include only a subset of times from one's network series, one can alter the `times` argument:

```
> samp.fit.2 <- stergm(samp,
   formation= ~edges+mutual+cyclicalties+transitiveties,
         dissolution = ~edges+mutual+cyclicalties+transitiveties,
         estimate = "CMLE",
   times=1:2
         )
```

How do these coefficients compare?

7 Example 2: One network cross section and durational information

Now, let us imagine a different scenario in which we have observed two types of data: a single cross-sectional network, and a mean relational duration. Let us say the cross-sectional network is `flobusiness`, and the mean relational duration we have witnessed is 10 time steps. Furthermore, we are willing (for reasons of theory or convenience) to assume a purely homogeneous dissolution process (that is, every existing relationship has the same probability of dissolving as all others, and at all times). For a cross-sectional ERGM, a purely homogeneous model is one with just a single term in it for an edge count. The same is true for either of the two formulas in a STERGM.

The steps we will go through are:

1. Specify formation and dissolution models (`formation` and `dissolution`).

   We will begin by assuming a formation model identical to the model we fit in the cross-sectional case:

   ```
   formation = ~edges+gwesp(0,fixed=T)
   ```

   Analogously to cross-sectional ERGMs, our assumption of completely homogeneous dissolution corresponds to a model with only an edge-count term in it. In STERGM notation this is:

   ```
   dissolution = ~edges
   ```

   which correspond to the probability statement:

   $$\ln \frac{P(Y_{ij,t+1} = 1 \mid Y_{ij,t} = 1)}{P(Y_{ij,t+1} = 0 \mid Y_{ij,t} = 1)} = \theta * \delta(y) \tag{6}$$

   where the one term in the $\delta(y)$ vector is the edge count of the network.

2. Calculate `theta.diss`.

   Our dissolution model is applied to the set of ties that exist at any given time point, as reflected in the conditional present in both the numerator and denominator of Equation (8). The numerator thus represents the case where a tie persists to the next step, and the denomiator represents the case where it dissolves. Furthermore, $\delta(y_{ij}) = 1$

for all i, j for the case of the edge count statistic. We define the probability of persistance from one time step to the next for actor pair i, j as p_{ij}, and the probability of dissolution as $q_{ij} = 1 - p_{ij}$. Our dissolution model is Bernoulli; that is, all edges have the same probability of dissolution, and thus of persistence, so we further define $p_{ij} = p \forall i, j$ and $q_{ij} = q \forall i, j$. Then:

$$\ln\left(\frac{p_{ij}}{1-p_{ij}}\right) = \theta * \delta(y_{ij})$$

$$\ln\left(\frac{p}{1-p}\right) = \theta$$

$$\ln\left(\frac{1-q}{q}\right) = \theta$$

$$\ln\left(\frac{1}{q} - 1\right) = \theta$$

And because this is a discrete memoryless process, durations are geometric; symbolizing mean relational duration as d, we have $d = \frac{1}{q}$, and thus:

$$\theta = \ln(d-1) \tag{7}$$

So, for our dissolution model, theta.diss $= \ln(10-1) = \ln 9 = 2.197$:

```
> theta.diss <- log(9)
```

In short, because our dissolution model is dyadic independent, we can calculate it using a (rather simple) closed form solution.

3. Decide upon our targets. For cross-sectional ERGMs, the model is by default fit using the sufficient statistics present in the cross-sectional network. For the STERGM with two cross-sections that we fit above, we also wanted to use the sufficient statistics present in the two observed networks. In the case of one cross-section but a formation and dissolution model, the reasonable default is less clear. Thus, the user is required to supply this information via the `targets` argument. This can take a one-sided formula listing the terms to be fit; or, if the formula is identical to either the formation or dissolution model, the user can simply pass the string `"formation"` or `"dissolution"`, respectively. If one is specifying `targets="formation"`, dissolution should be an offset, and vice versa. If the values to be targeted for those terms are anything other than the sufficient statistics present in

the cross-sectional network, then those values can be passed with the argument `target.stats`. In this case, we want to use the sufficient statistics present in the cross-sectional network for the model terms in the formation model.

4. Estimate the formation model, conditional on the dissolution model. We put it all together for our first call to `stergm`, adding in one additional control argument that helps immensely with monitoring model convergence (and is just plain cool): plotting the progress of the coefficient estimates and the simulated sufficient statistics in real time. When one is using a GUI tool like RStudio, it helps to output this plotting to a separate window, which we do below with the function `X11` (and then turn off that window again with `dev.off()`):

   ```
   > X11()
   > stergm.fit.1 <- stergm(flobusiness,
             formation= ~edges+gwesp(0,fixed=T),
             dissolution = ~offset(edges),
             targets="formation",
             offset.coef.diss = theta.diss,
             estimate = "EGMME",
             control=control.stergm(SA.plot.progress=TRUE)
             )
   > dev.off()
   ```

   ```
   Iteration 1 of at most 20:
   ```
   ⇒ Lots of output snipped. ⇐
   ```
   == Phase 3:  Simulate from the fit and estimate standard errors.==
   ```

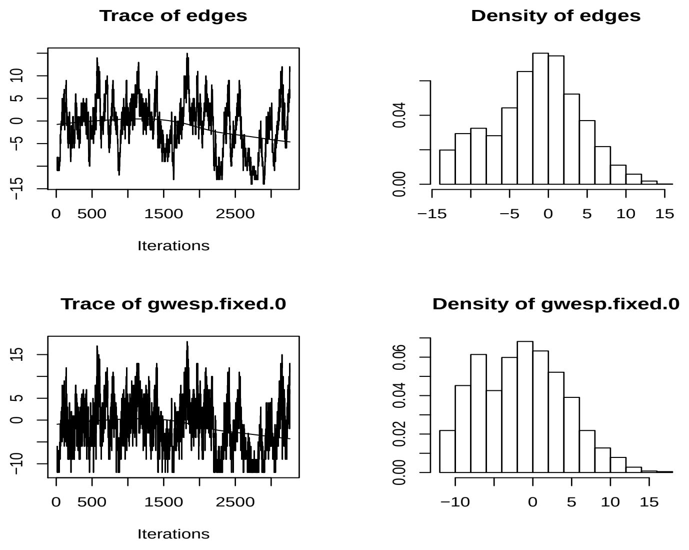
   First, we should double-check to make sure the fitting went well:

   ```
   > mcmc.diagnostics(stergm.fit.1)
   ```

   ```
   ==========================
   EGMME diagnostics
   ==========================
   ```

   ⇒ Lots of output snipped. ⇐

   Since those look good, we can next query the object in a variety of ways to see what we have:



```
> stergm.fit.1

Formation Coefficients:
        edges  gwesp.fixed.0
       -6.540          2.358
Dissolution Coefficients:
edges
2.197

> names(stergm.fit.1)

 [1] "network"         "formation"       "dissolution"
 [4] "targets"         "target.stats"    "estimate"
 [7] "covar"           "opt.history"     "sample"
[10] "sample.obs"      "control"         "reference"
[13] "mc.se"           "constraints"     "formation.fit"
[16] "dissolution.fit"

> stergm.fit.1$formation

~edges + gwesp(0, fixed = T)

> stergm.fit.1$formation.fit
```

```
EGMME Coefficients:
        edges  gwesp.fixed.0
       -6.540          2.358

> summary(stergm.fit.1)

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

Formula:   ~edges + gwesp(0, fixed = T)

Iterations:  NA

Equilibrium Generalized Method of Moments Results:
              Estimate Std. Error MCMC % p-value
edges          -6.5396     1.1178      0 < 1e-04 ***
gwesp.fixed.0   2.3582     0.8378      0 0.00572 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


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

Formula:   ~offset(edges)

Iterations:  NA

Equilibrium Generalized Method of Moments Results:
      Estimate Std. Error MCMC % p-value
edges    2.197         NA     NA      NA


 The following terms are fixed by offset and are not estimated:
  edges
```

We have now obtained estimates for the coefficients of a formation model that, conditional on the stated dissolution model, yields simulated targets that matched those observed. Something very useful we have also gained in the process is the ability to simulate networks with the desired cross-sectional structure and mean relational duration. This ability serves us well for any application areas that requires us to simulate phenomena on dynamic networks, whether they entail the diffusion of information or disease, or some other process.

```
> stergm.sim.1 <- simulate.stergm(stergm.fit.1, nsim=1,
     time.slices = 1000)
```

Understanding this object requires us to learn about an additional piece of *statnet* functionality: the *networkDynamic* package.

8 networkDynamic

In *statnet*, cross-sectional networks are stored using objects of class *network*. Tools to create, edit, and query network objects are in the package *network*. Dynamic networks are now stored as objects with two classes (*network* and *networkDynamic*). They can thus be edited or queried using standard functions from the *network* package, or using additional functions tailored specifically to the case of dynamic networks in the package *networkDynamic*.

To illustrate, let us begin with the network that we just created:

```
> stergm.sim.1

NetworkDynamic properties:
  distinct change times: 946
  maximal time range: -Inf to Inf

 Network attributes:
  vertices = 16
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 120
    missing edges= 0
    non-missing edges= 120
 Vertex attribute names:
    priorates totalties vertex.names wealth
```

We can deduce from the number of edges that this likely represents the cumulative network—that is, the union of all edges that exist at any point in time over the course of the simulation. What does the network look like at different time points? The function `network.extract` allows us to pull out the network at an instantanoues time point (with the argument `at`), or over any given spell (with the arguments `onset` and `terminus`).

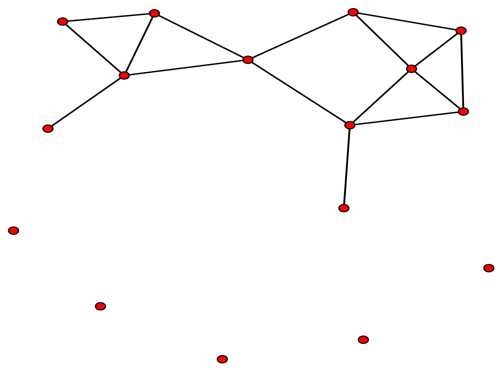
```
> net429 <- network.extract(stergm.sim.1,at=429)
> net429

NetworkDynamic properties:
  distinct change times: 473
  maximal time range: -Inf to Inf

 Network attributes:
  vertices = 16
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 26
    missing edges= 0
    non-missing edges= 26

 Vertex attribute names:
    priorates totalties vertex.names wealth
```

For any one of these time points, we can look at the network structure:

```
> plot(network.extract(stergm.sim.1,at=882))
```

How well do the cross-sectional networks within our simulated dynamic network fit the probability distribution implied by our model? We can check by considering the summary statistics for our observed network, and those for our cross-sectional networks.

```
> summary(flobusiness~edges+gwesp(0,fixed=T))
```



```
        edges gwesp.fixed.0
           15            12
```

> colMeans(attributes(stergm.sim.1)$stats)

```
        edges gwesp.fixed.0
       15.976        12.909
```

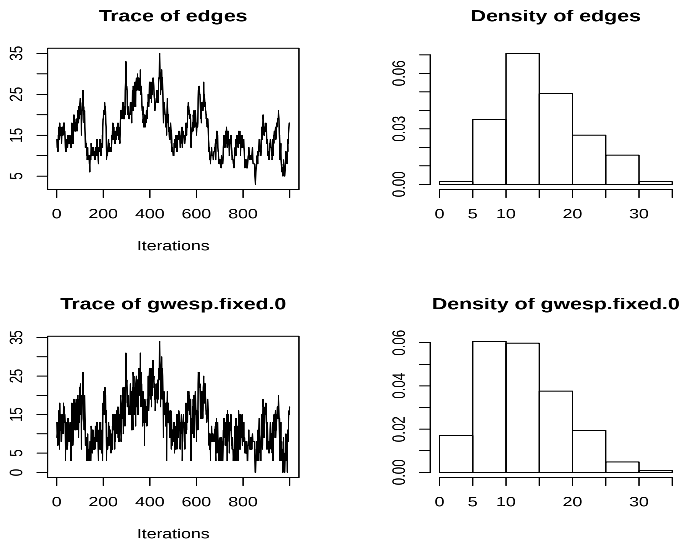
And we can also easily look at a time series and histogram for each statistic:

> plot(attributes(stergm.sim.1)$stats)

Or even at a scatter plot of the two network statistics for each simulated network cross-section, to see how these co-vary for this model:

> plot(as.matrix(attributes(stergm.sim.1)$stats))

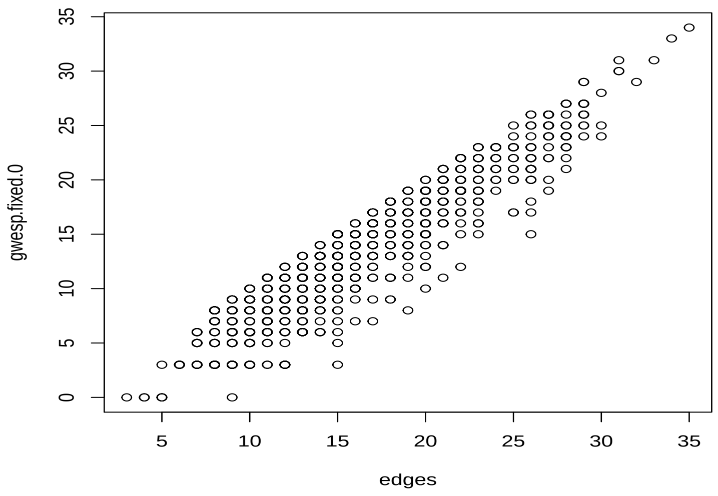
We should also check to make sure that our mean duration is what we expect (10 time steps). This requires knowing an additional function: `as.data.frame`, which, when applied to an object of class `networkDynamic`, generates a timed edgelist. Although right-censoring is present for some edges in our simulation, with a mean duration of 10 time steps and a simulation 1000 time steps long, its effect on our observed mean duration should be trivial.



```
> stergm.sim.1.dm <- as.data.frame(stergm.sim.1)
> names(stergm.sim.1.dm)

[1] "onset"             "terminus"
[3] "tail"              "head"
[5] "onset.censored"    "terminus.censored"
[7] "duration"          "edge.id"

> mean(stergm.sim.1.dm$duration)

[1] 9.853794
```

The information on when an edge is active and when it is inactive is stored within our `network` object as the edge attribute `active`. Vertices, too, are capable of becoming active and inactive within `networkDynamic`, and this information is stored as a vertex attribute. Most of the time, users should access this information indirectly, through functions like `network.extract` or `as.data.frame`. Additional functions to query or set activity include `is.active`, `activate.vertex`, `deactivate.vertex`, `activate.edge`, and `deactivate.edge`, all documented in `help(package="networkDynamic")`.

For those who want to look under the hood, they can see the activity spells directly. For a single edge, say, edge number 25, use:



```
> get.edge.activity(stergm.sim.1, 25)

[[1]]
     [,1] [,2]
[1,]    9   10
[2,]  221  225
[3,]  227  253
[4,]  342  343
[5,]  444  458
[6,]  474  483
[7,]  546  575
[8,]  628  638
```

or, with more info:

```
> get.edge.activity(stergm.sim.1, 25, as.spellList=T)

    onset terminus tail head onset.censored
354     9       10    3   10          FALSE
355   221      225    3   10          FALSE
356   227      253    3   10          FALSE
357   342      343    3   10          FALSE
```

```
358   444       458     3   10           FALSE
359   474       483     3   10           FALSE
360   546       575     3   10           FALSE
361   628       638     3   10           FALSE
    terminus.censored duration edge.id
354             FALSE        1      25
355             FALSE        4      25
356             FALSE       26      25
357             FALSE        1      25
358             FALSE       14      25
359             FALSE        9      25
360             FALSE       29      25
361             FALSE       10      25
```

Note that `networkDynamic` stores spells in the form [onset,terminus), meaning that the spell is inclusive of the onset and exclusive of the terminus. So a spell of 3,7 means the edge begins at time point 3 and ends just before time point 7. networkDynamic can handle continuous-time spell information. However, since STERGMs are discrete-time with integer steps, what this means for STERGM is that the edge is inactive up through time step 2; active at time steps 3, 4, 5, and 6; and inactive again at time step 7 and on. Its duration is thus 4 time steps.

9 Showing off your dynamic network with ndtv

We can visualize the simulated dynamic network `stergm.sim.1` using the `ndtv` package, which animates the transitions between each time step of the network. Install and load the package:

```
> install.packages('ndtv',repos="http://statnet.csde.washington.edu")
> library(ndtv)
```

Setup instructions for the ndtv package can be found in the `ndtv` vignette.

Some of the parameters for rendering the animation are shown below. `tween.frames` specifies the speed of the transition between timesteps (lower is faster, but more jittery; the default is 10). `show.time` and `show.stats` specify what captions to display along with the plot.

```
> render.par=list(tween.frames=5,show.time=T,
                  show.stats="~edges+gwesp(0,fixed=T)")
```

You can vary the size and color of both the vertices and edges based on their attributes. Here, we plot larger vertices for wealthier families.

```
> wealthsize = log(get.vertex.attribute(flobusiness, "wealth")) * 2/3
```

The simulation results `stergm.sim.1` can be animated using the following commands. A graphics window will pop out and show the rendering progress. The command ani.replay() can be used to replay the animation.

```
> render.animation(stergm.sim.1,render.par=render.par,
                   edge.col="darkgray",displaylabels=T,
                   label.cex=.8,label.col="blue",
                   vertex.cex=wealthsize)
> x11()
> ani.replay()
```

The animation is stored in the R environment in a hidden variable; it can be replayed at any time using the `ani.replay` function. However, each time the `render.animation` function is called, that variable is overridden. To share the video (on YouTube, for example), the animation can be saved as an MP4 movie file using the following commands. The codec `ffmpeg` must be installed. The size and quality of the movie file can be adjusted using the `ani.width`, `ani.height`, and `-b` parameters.

```
> ani.options(ffmpeg='C:\\ffmpeg\\bin\\ffmpeg.exe')
> ani.options(outdir='C:\\statnet\\ndtvdemo')
> saveVideo(ani.replay(),video.name="stergm.sim.1.mp4",
           other.opts="-b 5000k",clean=TRUE,
           ani.width=900, ani.height=768)
```

Time slices and aggregation

We can use the parameter `slice.par` in the compute.animation function to do some optional pre-processing of the networkDynamic object before rendering. `start` and `end` lets us animate a time segment of the dynamic network. Because the animation process may take a long time on larger networks, it is useful to peek at the layout for a short time segment before rendering the whole simulation.

For networks that change very slowly, we can set a larger time step size using parameter `interval`, and aggregate the edges over that interval using aggregate.dur. The rule "any" specifies that an edge that's active at any time step during the aggregation interval will show up in the plot.

Most of the time, the interval should be equal to the aggregate.dur. In the following example of classroom interactions, we kept the time step interval at 1, but aggregated over 5 time intervals, so that the time slices overlap and reveal slower structural patterns in the network. Overlapping the aggregation windows provides a smoothing effect, similar to a moving average, but the network density will be inaccurately represented. Compare the version below with the one with default aggregation settings.

```
> # directly animate a networkDynamic object, defining time slices
> data(McFarland_cls33_10_16_96)
> slice.par<-list(start=0,end=30,interval=1,
                  aggregate.dur=5,rule="any")
> compute.animation(cls33_10_16_96,
                    slice.par=slice.par,animation.mode='MDSJ')
> cols = rep('gray', 20); cols[7] = 'red'
> x11()
> render.animation(cls33_10_16_96, displaylabels=FALSE,
                  vertex.cex=1.5, vertex.col=cols)
> ani.replay()
```

10 Independence within and across time steps

STERGMs assume that the formation and dissolution processes are indepedent of each other *within the the same time step.*

This does not necessarily mean that they will be independent across time. In fact, for any dyadic dependent model, they will not. To see this point, think of a romantic relationship example with:

```
formation = ~edges+degree(2:10)
dissolution = ~edges
```

with increasingly negative parameters on the degree terms. What this means is that there is some underlying tendency for relational formation to occur, which is considerably reduced with each pre-existing tie that the two actors involved are already in. In other words, there is a strong prohibition against being in multiple simultaneous romantic relationships. However, dissolution is fully independent—all existing relationships have the same underlying dissolution probability at every time step. (The latter assumption is probably unrealistic; in practice, if someone just acquired a second partner, their first is likely to be at increased risk of dissoving their relation. We ignore this now for simplicity).

Imagine that Chris and Pat are in a relationship at time `t`. During the time period between `t` and `t+1`, whether they break up does not depend on when either of them acquires a new partner, and vice versa. Let us assume that they do *not* break up during this time. Now, during the time period between `t+1` and `t+2`, whether or not they break up is dependent on the state of the network at time `t+1`, and that depends on whether either of them they acquired new partners between `t` and `t+1`.

The simple implication of this is that in this framework, formation and dissolution can be dependent, but that dependence occurs in subsequent time steps, not simultaneously.

Note that a time step here is abritrary, and left to the user to define. One reason to select a smaller time interval is that it makes this assumption more justifiable. With a time step of 1 month, then if I start a new relationship today, the earliest I can break up with my first partner as a direct result of that new partnership is in one month. If my time step is a day, then it is in 1 day; the latter is likely much more reasonable. The tradeoff is that a shorter time interval means longer computation time for both model estimation and simulation, as will be seen below. You will see throughout this talk that there are multiple positives and negatives to having a short time step and having a long time step.

At the limit, this can in practice approximate a continuous-time model—the only issue is computational limitations.

11 Example 3: Approximation with long durations

For the type of model we saw in Example 2 (with a known dissolution model that contains a subset of terms from the formation model), it can be shown that a good set of starting values for the estimation of the formation model are as follows: (1) fit the terms in the formation model as a static ERGM on the cross-sectional network; and (2) subtract the values of the dissolution parameters from the corresponding values in the cross-sectional model. The result is a vector of parameter values that form a reasonable place to start the MCMC chain for the estimation of the formation model. This is in fact exactly what the `stergm` estimation code does by default for this type of model.

When mean relational duration is very long, this approximation is so good that it may not be necessary to run a STERGM estimation at all. Especially if your purpose is mainly for simulation, the approximation may be all you need. This is a very useful finding, since models with long mean duration are precisely the ones that are the slowest and most difficult to fit using EGMME. That's because, with long durations, very few ties will change between one time step and another, giving the fitting algorithm very little information on which to perform the estimation.

Of course, in order to be able to take advantage of this method, it is necessary for the terms in your dissolution model to be a subset of the terms in your formation model.

To illustrate, let us reconsider Example 2, with a mean relational duration of 100 time steps.

```
> theta.diss.100 <- log(99)
```

First, we treat the formation process as if it were a stand-alone cross-sectional model, and estimate it using a standard cross-sectional ERGM. We did, in fact, fit this cross-sectional model earlier:

```
> summary(fit1)

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

Formula:   flobusiness ~ edges + gwesp(0, fixed = T)

Iterations:  20

Monte Carlo MLE Results:
              Estimate Std. Error MCMC % p-value
edges          -3.3674     0.6128      0 < 1e-04 ***
gwesp.fixed.0   1.5646     0.5810      0 0.00812 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

     Null Deviance: 166.36  on 120  degrees of freedom
 Residual Deviance:  78.21  on 118  degrees of freedom

AIC: 82.21    BIC: 87.79    (Smaller is better.)
```

```
> theta.form <- fit1$coef
> theta.form

        edges gwesp.fixed.0
    -3.367383      1.564649

>
```

Then, we subtract the values of the dissolution θ from each of the corresponding values in the formation model. In this example, the dissolution model contains only an edges term, so this coefficient should be subtracted from the starting value for the edges term in the formation model.

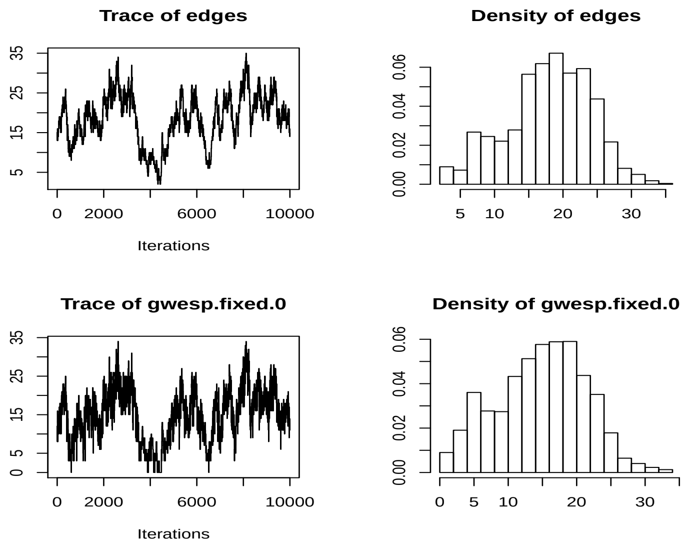
```
> theta.form[1] <- theta.form[1] - theta.diss.100
```

How well does this approximation do in capturing our desired dynamic network properties? First, we can simulate from it:

```
> stergm.sim.2 <- simulate(flobusiness,
          formation=~edges+gwesp(0,fixed=T),
          dissolution=~edges,
          monitor="all",
          coef.form=theta.form,
          coef.diss=theta.diss.100,
          time.slices=10000)
```

Then check the results in terms of cross-sectional network structure and mean relational duration?

```
> summary(flobusiness~edges+gwesp(0,fixed=T))

        edges gwesp.fixed.0
           15            12

> colMeans(attributes(stergm.sim.2)$stats)

        edges gwesp.fixed.0
      18.4236       15.2605

> stergm.sim.dm.2 <- as.data.frame(stergm.sim.2)
> mean(stergm.sim.dm.2$duration)

[1] 103.912

> plot(attributes(stergm.sim.2)$stats)
```



12 Example 4: Simulation driven by egocentric data

In many cases, people's primary interest in using dynamic networks is to simulate some diffusion process on one or more networks with similar features. Increasingly, our knowledge about those features come in the form of egocentrically sampled data, not from the traditional network census in a bounded population. Both *ergm* and *stergm* have methods for handling these situations.

For example, imagine that you want to model HIV transmission among a population of gay men in steady partnerships. 50% of the men are White and 50% are Black. You collect egocentric partnership data from a random (ha! ha!) sample of these men. Your data say:

1. There are no significant differences in the distribution of momentary degree (the number of ongoing partnerships at one point in time) reported by White vs. Black men. The mean is 0.90, and the overall distribution is:
   (a) 36% degree 0
   (b) 46% degree 1
   (c) 18% degree 2+

2. 83.3% of relationships are racially homogeneous

We also have data (from these same men, or elsewhere) that tell us that the mean duration for a racially homogenous relationship is 10 months, while for a racially mixed one it is 20 months. (Perhaps this is because the social pressure against cross-race ties makes it such that those who are willing to enter them are a select group, more committed to their relationships).

Before we model the disease transmission, we need a dynamic network that possesses each of these features to simulate it on.

Our first step is to create a 500-node undirected network, and assign the first 250 nodes to race 0 and the second to race 1. The choice of 500 nodes is arbitary.

```
> msm.net <- network.initialize(500, directed=F)
> msm.net %v% 'race' <- c(rep(0,250),rep(1,250))
> msm.net

 Network attributes:
  vertices = 500
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 0
    missing edges= 0
    non-missing edges= 0

 Vertex attribute names:
    race vertex.names
```

ERGM and STERGM have functionality that allow us to simply state what the target statistics are that we want to match; we do not actually need to generate a network that has them. The formation formula and target statistics that we need are:

```
> msm.form.formula <- ~edges+nodematch('race')+degree(0)+
     concurrent
> msm.target.stats <- c(225,187,180,90)
```

Why don't we specify `degree(1)` as well? How did we get those values?

Now let us turn to dissolution. We are back to the case where we can solve these explicitly, although this is complicated slightly by the fact that our dissolution probabilities differ by the race composition of the members. One dissolution formula for representing this is:

```
> msm.diss.formula <- ~offset(edges)+offset(nodematch("race"))
```

These two model statistics means that there will be two model coefficients. Let us call them θ_1 and θ_2 for the edges and nodematch terms, respectively. Let us also refer to the change statistics for actor pair i, j for each of these as $\delta_1(y_{ij})$ and $\delta_2(y_{ij})$, respectively.

Thus the log-odds expression for dissolution that we saw earlier would here be expressed as:

$$\ln \frac{P(Y_{ij,t+1} = 1 \mid Y_{ij,t} = 1)}{P(Y_{ij,t+1} = 0 \mid Y_{ij,t} = 1)} = \theta_1 \delta_1(y_{ij}) + \theta_2 \delta_2(y_{ij}) \tag{8}$$

Note that $\delta_1(y_{ij})$ would equal 1 for all actor pairs, while $\delta_2(y_{ij})$ would equal 1 for race homophilous pairs and 0 for others. That means that the log-odds of tie persistence will equal θ_1 for mixed-race couples and $\theta_1 + \theta_2$ for race-homophilous couples. This suggests that we should be able to calculate θ_1 directly, and subsequently calculate θ_2.

Following the logic we saw in Example 2, we can see that:

$$\theta_1 = \ln d_{mixed} - 1 \tag{9}$$

and therefore $\theta_1 = \ln(20 - 1) = \ln 19 = 2.944$.

Furthermore,

$$\theta_1 + \theta_2 = \ln d_{homoph} - 1 \tag{10}$$

and therefore $\theta_2 = \ln(d_{homoph} - 1) - \theta_1 = \ln(10 - 1) - 2.944 = -0.747$.

So, we have:

```
> msm.theta.diss <- c(2.944, -0.747)
```

We add in one additional control parameter—`SA.init.gain`—giving it a small value (the default is 0.1). As the help page for `control.stergm` sagely advises, "If the process initially goes crazy beyond recovery, lower this value." This slows down estimation, but also makes it more stable. From trial and error, we know that this model, fit to this relatively large network, does better with this smaller value.

Putting it all together (including the syntax to control the window for real-time monitoring) gives us:

```
> set.seed(0)
> X11()
> msm.fit <- stergm(msm.net,
          formation= msm.form.formula,
          dissolution= msm.diss.formula,
          targets="formation",
          target.stats= msm.target.stats,
          offset.coef.diss = msm.theta.diss,
          estimate = "EGMME",
          control=control.stergm(SA.plot.progress=TRUE,
              SA.init.gain=0.005)
 )
> dev.off()
```

```
   Iteration 1 of at most 20:
```

⇒ Lots of output snipped. ⇐

```
======== Phase 3:  Simulate from the fit and estimate standard errors.
========
```

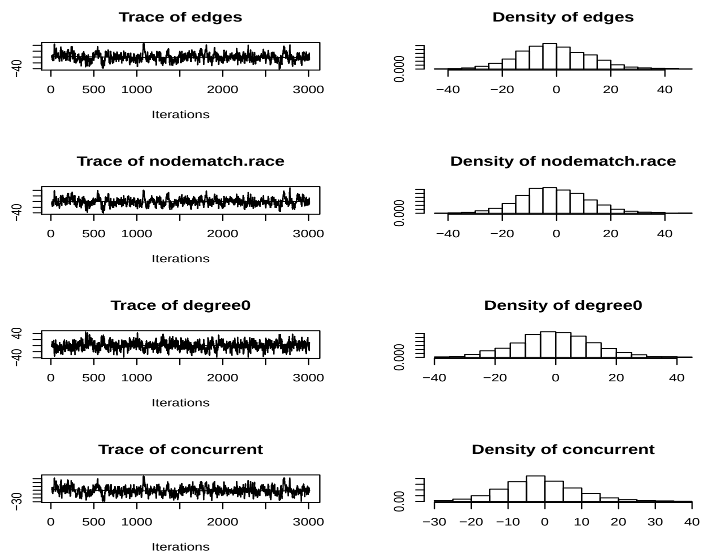
Let's first check to make sure it fit well:

```
> mcmc.diagnostics(msm.fit)
```

⇒ Lots of output snipped. ⇐

and see what the results tell us:

```
> summary(msm.fit)

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

Formula:   ~edges + nodematch("race") + degree(0) + concurrent

Iterations:  NA

Equilibrium Generalized Method of Moments Results:
               Estimate Std. Error MCMC % p-value
edges          -9.96387    0.36493      0  <1e-04 ***
nodematch.race  2.30755    0.19525      0  <1e-04 ***
```



```
degree0          -0.19519    0.08554       0  0.0225 *
concurrent       -0.82986    0.11630       0  <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


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

Formula:   ~offset(edges) + offset(nodematch("race"))

Iterations:  NA

Equilibrium Generalized Method of Moments Results:
               Estimate Std. Error MCMC % p-value
edges             2.944         NA     NA      NA
nodematch.race   -0.747         NA     NA      NA


 The following terms are fixed by offset and are not estimated:
  edges nodematch.race
```

Now, we simulate a dynamic network:

```
> msm.sim <- simulate(msm.fit,time.slices=1000)
```

and compare the outputs to what we expect, in terms of cross-sectional structure:

```
> colMeans(attributes(msm.sim)$stats)

         edges nodematch.race        degree0     concurrent
       226.287        187.203        179.847         91.017

> msm.target.stats

[1] 225 187 180  90
```

And relationship length:

```
> msm.sim.dm <- as.data.frame(msm.sim)
> names(msm.sim.dm)

[1] "onset"              "terminus"
[3] "tail"               "head"
[5] "onset.censored"     "terminus.censored"
[7] "duration"           "edge.id"

> msm.sim.dm$race1 <- msm.sim.dm$head>250
> msm.sim.dm$race2 <- msm.sim.dm$tail>250
> msm.sim.dm$homoph <- msm.sim.dm$race1 == msm.sim.dm$race2
> mean(msm.sim.dm$duration[msm.sim.dm$homoph==T])

[1] 9.861967

> mean(msm.sim.dm$duration[msm.sim.dm$homoph==F])

[1] 20.57526
```

You can also check to see how much this estimate changes when you exclude the edges that are censored:

```
> mean(msm.sim.dm$duration[msm.sim.dm$homoph==T &
   msm.sim.dm$onset.censored==F & msm.sim.dm$terminus.censored==F ])

[1] 9.876734

> mean(msm.sim.dm$duration[msm.sim.dm$homoph==F &
   msm.sim.dm$onset.censored==F & msm.sim.dm$terminus.censored==F ])

[1] 20.43644
```

13 Temporally extended attributes

One of the new features in networkDynamic is the ability to track edge or vertex attributes that change with time. Some examples of these temporally extended attributes (TEA) are age and disease status. If a network has TEAs defined, they can be used to control graphic properties of the network which change over time. The examples below show a disease spreading through two networks with very different behaviors.

The first dynamic network is a large (10,000 people) simulation of relationships in Botswana. The infection simulation begins with 5 seed infectors, which are designated by the infected attribute with the function `activate.vertex.attribute`. At each time step, those who are infected can be determined with the function `get.vertex.attribute.active`. For each infected person, the partners can be found by using `get.neighborhood.active`. Each partner who is not already infected has a 20% chance of becoming infected (a very simple model). Again, infection status can be updated using `activate.vertex.attribute`.

```
> load("Bots_10Kobs_fit.Rdata")
> seeds = c(8796,7460,403,1459,6307)
> infectsim = function(dnet, duration, seeds=NULL, infection.prob=0.2) {
   if (is.null(seeds)) seeds = sample(1:network.size(dnet), 5)

   activate.vertex.attribute(dnet,
          'infected', 'gray',onset=0,terminus=Inf)
   activate.vertex.attribute(dnet,
          'infected', 'red', onset=0, terminus=Inf, v=seeds)

   transmissions = NULL
   for (t in 1:duration) {
     infecteds = which(get.vertex.attribute.active(
       dnet, prefix='infected', at=t) == 'red')

     newinfects = NULL
     for (i in infecteds) {
       partners = get.neighborhood.active(dnet, v=i, at=t)
       # discordant couples
       partners = partners[!(partners %in% infecteds)]
       partners = partners[!(partners %in% newinfects)]
       for (j in partners) {
         if (runif(1) < infection.prob) {
           newinfects = c(newinfects, j)
           # record transmission
           transmissions = rbind(transmissions, c(i, j))
         }
       }
     }

     if (!is.null(newinfects))
       activate.vertex.attribute(dnet,
             'infected', 'red', onset=t, terminus=Inf, v=newinfects)
     cat(t, ' ')
   }
   set.network.attribute(dnet, 'transmissions', transmissions)
   return(dnet)
 }
> dnet = infectsim(dnet, duration=1000, seeds)
```

Since the network is too large to be properly visualized, we will extract the sub-network of only the people who were infected, using the function `get.inducedSubgraph`.

```
> allinfected = which(get.vertex.attribute.active(
  dnet, prefix='infected', at=1000) == 'red')
> subnet2 <-get.inducedSubgraph(dnet, allinfected)
```

In the animation here, we use both a standard vertex attribute (sex), and a temporally extended attribute (infected) to control the shape and color of the vertices. If an attribute name is in quotes ("infected"), the renderer will attempt to match it to a TEA. Note that the infected attribute is set to either “red” for infected, or “gray” for uninfected. Currently this is the simplest way to specify the color representation of a TEA; there will be a function to remap attributes to customized graphical properties in a future version of ndtv.

```
> slice.par <- list(start=0, end=1000,
                   interval=10, aggregate.dur=10, rule='any')
> compute.animation(subnet2,
                   slice.par=slice.par,animation.mode='MDSJ')
> x11()
> sex.shapes <- ifelse(
   get.vertex.attribute(subnet2, 'sex')=="male", 4, 50)
> render.animation(subnet2,
                  displaylabels=FALSE,vertex.cex=.8,
                  vertex.col="infected",
                  edge.col='#77777780', vertex.sides=sex.shapes,
                  vertex.rot=45)
```

The dynamic network shown here represents mostly long-term heterosexual relationships, with some concurrency. Even when the animation is sped up to 10 time steps per frame, the number of relationship formations and dissolutions remain low. Transmission in a discordant relationship here is a near certainty.

In contrast, the MSM network simulated in Example 4 of this tutorial has much shorter relationship durations. The same infection simulation on that network can be visualized and compared. Here, the borders of the vertices are colored by race.

```
> subnet2 <- infectsim(msm.sim, duration=120)
> slice.par <- list(start=0, end=100,
                   interval=1, aggregate.dur=1, rule='any')
> compute.animation(subnet2,
                   slice.par=slice.par,animation.mode='MDSJ')
> x11()
> racecols <- ifelse(
   get.vertex.attribute(subnet2, 'race')==0, '#3C2E28', '#A57E6E')
> render.animation(subnet2,
                  displaylabels=FALSE,vertex.cex=1,
                  vertex.border=racecols,
                  vertex.col='infected', edge.col='#77777780')
> ani.replay()
```

14 Additional functionality

Both the `stergm` functions and the `networkDynamic` package have additional functionality, which you can learn about and explore through the use of R's many help features. If you begin to use them in depth, you will likely have further questions. If so, we encourage you to join the statnet users' group (`http://csde.washington.edu/statnet/statnet_users_group.shtml`), where you can then post your questions (and possibly answer others). You may also encounter bugs; please use the same place to report them. Happy stergming!

References

[1] Carter T. Butts, Ayn Leslie-Cook, Pavel N. Krivitsky, and Skye Bender-deMoll. `networkDynamic`: Dynamic Extensions for Network Objects. The Statnet Project (`http://www.statnet.org`), 2013. R package version 0.4. `http://CRAN.R-project.org/package=networkDynamic`

[2] Pavel N. Krivitsky and Mark S. Handcock. A Separable Model for Dynamic Networks. *Journal of the Royal Statistical Society, Series B*, 2012, In press. `http://arxiv.org/abs/1011.1937`

[3] Pavel N. Krivitsky. Modeling of Dynamic Networks based on Egocentric Data with Durational Information. Pennsylvania State University Department of Statistics, 2012(2012-01). `http://stat.psu.edu/research/technical-reports/2012-technical-reports`

[4] Pavel N. Krivitsky and Mark S. Handcock. `tergm`: Fit, Simulate and Diagnose Models for Network Evoluation based on Exponential-Family Random Graph Models. The Statnet Project (`http://www.statnet.org`), 2013. R package version 3.1-0. `http://CRAN.R-project.org/package=tergm`