Modeling Relational Event Dynamics with statnet

2016 Sunbelt Social Networks Conference
Newport Beach, CA
April 5, 2016

Presenters:
Carter T. Butts (University of California, Irvine)
Christopher S. Marcum (US National Institutes of Health)

Assistants:
C. Ben Gibson (University of California, Irvine)
Francis Lee (University of California, Irvine)
Nolan Philips (University of California, Irvine)
Emma Smith (University of California, Irvine)
Fan Yin (University of California, Irvine)
Yue Yu (University of California, Irvine)
Xuhong Zhang (University of California, Irvine)

statnet Core Development Team:
Skye Bender-deMoll (University of Washington)
Carter T. Butts (University of California, Irvine)
Steven M. Goodreau (University of Washington)
Mark S. Handcock (University of California, Los Angeles)
David R. Hunter (Penn State University)
Pavel Krivitsky (University of Wollongong)
Martina Morris (University of Washington)

Table of contents **Author**

Basic resources

R webpage: http://www.r-project.org
Helpful **R** tutorials: http://cran.r-project.org/other-docs.html
statnet webpage: http://statnet.org
statnet help: statnet_help@statnet.org
Workshop web site: https://statnet.csde.washington.edu/trac/wiki/Sunbelt2015

Typographical conventions

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

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

All other text represents instructions and guidance.

SECTION 0. GETTING STARTED

**** The instructions in Section 0 match those on the workshop web page (https://statnet.csde.washington.edu/trac/wiki/Sunbelt2016). If you have already installed the required software, there is no need to do so again. ****

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

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

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

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

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

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

```
update_packages("statnet")
```

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

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

```
library(statnet)
```

0.3. Download and install supplemental packages:

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

```
install.packages("relevent")
install.packages("informR")
```

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

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

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

SECTION 1. DYADIC RELATIONAL EVENT MODELS WITH rem.dyad: ORDINAL TIMING

Dyadic relational event models are intended to capture the behavior of systems in which individual social units (persons, organizations, animals, etc.) direct discrete actions towards other individuals in their environment. Within the *relevent* package, the `rem.dyad` function is the primary workhorse for modeling dyadic data. Although less flexible than `rem` (discussed in sections 3 and 4), `rem.dyad` contains many features that make it easier to work with in the dyadic case.

Data for use with `rem.dyad` consists of dynamic edge lists, each edge being characterized by a sender, a recipient, and an event time. (Currently, self-edges and undirected edges are not supported -- this will change in future versions!) Ideally, event times are known exactly; however, under the piecewise constant hazard assumption (per Butts, 2008) the relational event family can still be identified up to a pacing constant so long as the order of events is known. Since the case of ordinal timing is somewhat simpler than that of exact timing, we consider it first.

1.0 Loading the relevent package, and the Sunbelt data

```
library(relevent)                          #Load the relevent library
load("relevent_sunbelt_2015.Rdata")        #Load the Sunbelt data - may need to change directory!
```

1.1 Getting a look at the WTC Police radio data

The data we will use here comes from the World Trade Center radio communication data set coded by Butts et al. (2007). It consists of radio calls among 37 named communicants belonging to a police unit at the World Trade Center complex on the morning of 9/11/2001. The edgelist is contained in an object called `WTCPoliceCalls`; printing it should yield output like the following:

```
WTCPoliceCalls
```

```
  number source recipient
1      1     16        32
2      2     32        16
3      3     16        32
4      4     16        32
5      5     11        32
6      6     11        32
...    ...   ...       ...
```

Note the form of the data: a matrix with the timing information, source (numbered from 1 to 37), and recipient (again numbered from 1 to 37) for each event (i.e., radio call). It is important to note that the WTC radio data was coded from transcripts that lacked detailed timing information; we do not therefore know precisely when these calls were made. We do, however, know the *order* in which calls were made, and can use this to fit relational event models with `rem.dyad`.

Before analyzing the data, it is helpful to consider what it looks like in time aggregated form. The workshop-supplied helper function `as.sociomatrix.eventlist` is useful for this purpose: it converts an event list into a valued sociomatrix, of the form used by other `statnet` routines. Let's convert the data to sociomatrix form, and visualize it using the `gplot` function of the *sna* package:

```
WTCPoliceNet <- as.sociomatrix.eventlist(WTCPoliceCalls,37)
gplot(WTCPoliceNet,edge.lwd=WTCPoliceNet^0.75,vertex.col=2+WTCPoliceIsICR,vertex.cex=1.25)
```

In this visualization, we have scaled edge widths by communication volume -- clearly, some pairs interact much more than others. Note also that we have colored vertices based on whether or not they occupy an institutionalized coordinative role (ICR), as indicated by the vector `WTCPoliceIsICR`. Those for whom this vector is `TRUE` (green) occupy roles within the police organization that would be expected to participate in coordinative activities; other actors were not identified as occupying such roles, based on the transcript data. In the analyses below, we will employ this covariate (as well as various endogenous mechanisms) to model the dynamics of radio communication within the WTC police network.

1.2 A first model: exploring ICR effects

Let's begin by fitting a very simple covariate model, in which the propensity of individuals to send and receive calls depends on whether they occupy institutionalized coordinative roles:

```
#First ICR effect - total interaction
```

```
wtcfit1<-rem.dyad(WTCPoliceCalls,n=37,effects=c("CovInt"),covar=list(CovInt=WTCPoliceIsICR),
    hessian=TRUE)
summary(wtcfit1)
```

Your output should look like this:

```
Relational Event Model (Ordinal Likelihood)

         Estimate  Std.Err Z value  Pr(>|z|)
CovInt.1 2.104464 0.069817  30.142 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Null deviance: 6921.048 on 481 degrees of freedom
Residual deviance: 6197.998 on 480 degrees of freedom
     Chi-square: 723.05 on 1 degrees of freedom, asymptotic p-value 0
AIC: 6199.998 AICC: 6200.007 BIC: 6204.174
```

The output gives us the covariate effect, as well as some uncertainty and goodness-of-fit information. The format is much like the output for a regression model, but coefficients should be interpreted per the relational event framework. In particular, the ICR role coefficient is the logged multiplier for the hazard of an event involving an ICR, versus a non-ICR event. (The effect is cumulative: an event in which one actor in an ICR calls another actor in an ICR gets twice the log increment.) We can see this impact in real terms as follows:

```
exp(wtcfit1$coef)           #Relative hazard for a non-ICR/ICR vs. a non-ICR/non-ICR event
exp(2*wtcfit1$coef)         #Relative hazard for an ICR/ICR vs. a non-ICR/non-ICR event
```

We have here considered a homogeneous effect of ICR status on sending and receiving; is it worth treating these effects separately? To do so, we enter the ICR covariate as a sender and receiver covariate (respectively):

```
wtcfit2<-rem.dyad(WTCPoliceCalls,n=37,effects=c("CovSnd","CovRec"),
    covar=list(CovSnd=WTCPoliceIsICR,CovRec=WTCPoliceIsICR),hessian=TRUE)
summary(wtcfit2)
```

Does the effect seem to differ? Let's see if fit improves (using the BIC):

```
wtcfit1$BIC-wtcfit2$BIC                     #Model 1 a bit lower - we prefer it
```

Model selection criteria are the preferred way to compare models, but one can also use a test of equality on the coefficients:

```
wtcfit2$coef                        #Extract the coefficients
wtcfit2$cov                         #Likewise, the posterior covariance matrix
#Heuristic Wald test of equality (not Bayesian, but whatever)
z<-diff(wtcfit2$coef)/sqrt(sum(diag(wtcfit2$cov))-2*wtcfit2$cov[1,2])
z
2*(1-pnorm(abs(z)))                 #Not conventionally significant - not strongly detectable
```

There might be some difference between the ICR sender and receiver effects, but they don't seem large enough to worry about. For now, we'll just stick with the simpler model (with a uniform effect on total interaction).

1.3 Bringing in endogenous social dynamics

One of the attractions of the relational event framework is its ability to capture endogenous social dynamics. In the following examples, we will examine several kinds of mechanisms that could conceivably impact communication among participants in the WTC police network. In each case, we first fit a candidate model, then compare that model to our best fitting model thus far identified. Where effects result in an improvement (as judged by the BIC), we include them in subsequent models.

To begin, we note that this is radio communication data. Radio communication is governed by strong conversational norms (in particular, radio SOP), which among other things mandate systematic turn-taking reciprocity. We can test for this via the use of *participation shifts*, particularly the AB-BA shift (a tendency for B to call A, given that A has just called B).

```
wtcfit3<-rem.dyad(WTCPoliceCalls,n=37,
    effects=c("CovInt","PSAB-BA"),covar=list(CovInt=WTCPoliceIsICR),hessian=TRUE)
summary(wtcfit3)                        #Looks like a strong effect...
wtcfit1$BIC-wtcfit3$BIC                 #We prefer model 3 to model 1 - reciprocity is in!
```

```
exp(wtcfit3$coef["PSAB-BA"])            #Reciprocating events are >1500 times as likely
```

What about other conversational norms? In general, we may expect that the current participants in an interaction may be likely to initiate the next call, a tendency that can also be captured with P-shift effects.

```
wtcfit4<-rem.dyad(WTCPoliceCalls,n=37,effects=c("CovInt","PSAB-BA","PSAB-BY","PSAB-AY"),
    covar=list(CovInt=WTCPoliceIsICR),hessian=TRUE)
summary(wtcfit4)                    #Seems like the effects are present, but let's test GOF...
wtcfit3$BIC-wtcfit4$BIC             #Yes, definite improvement
```

P-shift effects are "local," in that they depend only on the prior event. What about effects of recency (from the point of view of ego) on the tendency to send calls to others?

```
wtcfit5<-rem.dyad(WTCPoliceCalls,n=37,effects=c("CovInt","PSAB-BA","PSAB-BY",
    "PSAB-AY","RRecSnd","RSndSnd"),covar=list(CovInt=WTCPoliceIsICR),hessian=TRUE)
summary(wtcfit5)                    #Looks good; note that AB-BA is much smaller than before
wtcfit4$BIC-wtcfit5$BIC             #Substantial improvement
```

Finally, recall what our relational event data looked like when viewed in time-aggregated form. We observed a strongly hub-dominated network, with a few actors doing most of the communication. Could this be explained in part via a preferential attachment mechanism (per de Sola Price and others), in which those having the most air time become the most attractive targets for others to call? We can investigate this by including normalized total degree as a predictor of tendency to receive calls:

```
wtcfit6<-rem.dyad(WTCPoliceCalls,n=37,effects=c("CovInt","PSAB-BA","PSAB-BY",
    "PSAB-AY","RRecSnd","RSndSnd","NTDegRec"),covar=list(CovInt=WTCPoliceIsICR),hessian=TRUE)
summary(wtcfit6)                   #PA is drawing from recency, ICR effect, but not P-shifts
wtcfit5$BIC-wtcfit6$BIC            #Model is preferred
```

At this point, we've got a decent quorum of effects, and the deviance reduction is substantial. Of course, we could continue to investigate other mechanisms; see ?rem.dyad for the full range of options.

1.4 Assessing model adequacy

Model adequacy is an important consideration: even given that our model is the best of those we've seen, is it good enough for our purposes? There are many ways to assess model adequacy; here, we focus on the ability of the relational event model to predict the next event in the sequence, given those that have come before.

A natural question to ask when assessing the model is to ask when it is "surprised": when does it encounter observations that are relatively poorly predicted? To investigate this, we can examine the deviance residuals:

```
nullresid<- 2*log(37*36)   #What would be the deviance residual for the null?
hist(wtcfit6$residuals)    #Deviance residuals - most well-predicted, some around chance levels
abline(v=nullresid,col=2)

mean(wtcfit6$residuals<nullresid)   #Beating chance on almost all...
mean(wtcfit6$residuals<3)           #Upper limit of lower cluster is about 3
```

We seem to be doing pretty well here. As another way of evaluating the deviance residuals for the ordinal model, it is useful to note that the quantity exp(DR/2) (where DR is the deviance residual) is a "random guessing equivalent," or an effective number of events such that a random guess among such events as to which is coming next would be right as often as the model expects to be. We can easily compute this as follows:

```
quantile(exp(wtcfit6$residuals/2))              #"Random guessing equivalent" (ref is 1332)
```

Note that there are 1332 possible events, so we are doing much, much better than an uninformative baseline. Likewise, we've come a long way from our initial model:

```
quantile(exp(wtcfit1$residuals/2))                  #By comparison, first model much worse!
```

In addition to overall examination of residuals, it can be useful to ask which particular events seem to be sources of surprise:

```
cbind(WTCPoliceCalls,wtcfit6$residuals>nullresid)    #Which are the more surprising cases?
```

Using `as.sociomatrix.eventlist`, we can even pull out these events and view them in time-aggregated form. This can give us a better sense of the structural context in which they occur:

```
surprising<-as.sociomatrix.eventlist(WTCPoliceCalls[wtcfit6$residuals>nullresid,],37)
gplot(surprising)                                                    #Plot in network form

#Can also superimpose on the original network (coloring edges by fraction surprising)
edgecol<-matrix(rgb(surprising/(WTCPoliceNet+0.01),0,0),37,37)     #Color me surprised
gplot(WTCPoliceNet,edge.col=edgecol,edge.lwd=WTCPoliceNet^0.75,vertex.col=2+WTCPoliceIsICR)
```

Yet another approach to adequacy assessment is to consider the rank of the observed events in the predicted rate structure: that is, we ask to what extent the events viewed most likely to occur are in fact those that are observed.

```
hist(wtcfit6$observed.rank)                                  #Histogram of ranks
cbind(WTCPoliceCalls,wtcfit6$observed.rank)                  #Rank on a per-event basis (low is good)

#Sometimes useful to plot the ECDF of the observed ranks....
plot(ecdf(wtcfit6$observed.rank/(37*36)),
    xlab="Prediction Threshold (Fraction of Possible Events)",
    ylab="Fraction of Observed Events Covered",main="Classification Accuracy")
abline(v=c(0.05,0.1,0.25),col=2)
```

As the above indicates, we sometimes (in fact often) manage to get things exactly right: that is, the event predicted most likely to be the next in the sequence is in fact the one that is observed. Examining the match rate is a very strict notion of adequacy, but can be useful for assessing models that are strongly predictive.

```
wtcfit6$predicted.match                                     #Exactly correct src/target
mean(apply(wtcfit6$predicted.match,1,any))                  #Fraction for which something is right
mean(apply(wtcfit6$predicted.match,1,all))                  #Fraction entirely right
colMeans(wtcfit6$predicted.match)                           #Fraction src/target, respectively
```

Despite its simplicity, this model seems to fit extremely well. Further improvement is possible, but for many purposes we might view it as an adequate representation of the event dynamics in this WTC police network.

SECTION 2. DYADIC RELATIONAL EVENT MODELS WITH `rem.dyad`: EXACT TIMING

In the previous section, we considered dyadic relational event models in the case for which only ordinal timing information is available. We now proceed to the case of exact timing, in which we know the time at which each event occurs (relative to the onset of observation, which is treated as time 0).

2.0 The McFarland classroom data

For this section, we will make use of data collected by Dan McFarland (and published in McFarland and Bender-deMoll, 2006) on interaction among students and instructors within a high school classroom. (Note that the data employed here has been slightly modified from the original for illustrative purposes, in that small timing adjustments have been made to separate closely spaced events; those interested in using it for purposes other than practice are directed to the above paper in the *Journal of Social Structure*.) To see the event data itself, we may print it as follows:

```
Class
```

```
  StartTime FromId ToId
1     0.135     14   12
2     0.270     12   14
3     0.405     18   12
4     0.540     12   18
5     0.675      1   12
6     0.810     12    1
...     ...     ...  ...
691    50.910     17    6
692    50.920     NA   NA
```

As before, we have three columns: the event time, the event source (numbered from 1 to 20), and the event target (again, numbered 1 to 20). In this case, event time is given in increments of minutes from onset of observation. Note that the last row of the event list contains the time at which observation was terminated; it (and only it!) is allowed to contain NAs, since it has no meaning except to set the period during which events could have occurred. Where exact timing is used, the final entry in the edgelist is always interpreted in this way, and any source/target information on this row is ignored.

In addition to the `Class` edgelist, we also observe the covariates `ClassIsTeacher` (an indicator for instructor role) and `ClassIsFemale` (an indicator for gender). Visualizing the data in time-aggregate form gives us the following:

```
ClassNet<-as.sociomatrix.eventlist(Class,20)
gplot(ClassNet,vertex.col=4-2*ClassIsFemale,vertex.sides=3+ClassIsTeacher,vertex.cex=2,
    edge.lwd=ClassNet^0.75)
```


A dynamic visualization for this data is also available in the above-cited paper, and is well worth examining!

2.1 Modeling with covariates

We begin our investigation of classroom dynamics with a trivial intercept model, containing only a vector of 1s (`ClassIntercept`) as a sending effect:

```
classfit1<-rem.dyad(Class,n=20,effects=c("CovSnd"),covar=list(CovSnd=ClassIntercept),
    ordinal=FALSE,hessian=TRUE)
summary(classfit1)
```

Note that we must tell `rem.dyad` that we do not want to discard timing information (`ordinal=FALSE`). The model does not fit any better than the null because it is equivalent to the null model (but you must supply your own intercept, regardless!). As one would expect from first principles, this is really just an exponential waiting time model, calibrated to the observed communication rate:

```
(classfit1$m-1)/max(Class[,1])                  #Events per minute (on average)
20*19*exp(classfit1$coef)                       #Predicted events per minute (matches well!)
```

To make things more interesting, let's add effects for role and gender:

```
classfit2<-rem.dyad(Class,n=20,effects=c("CovSnd","CovRec"),
    covar=list(CovSnd=cbind(ClassIntercept,ClassIsTeacher,ClassIsFemale),
    CovRec=cbind(ClassIsTeacher,ClassIsFemale)),ordinal=FALSE,hessian=TRUE)
```

```
summary(classfit2)
classfit1$BIC-classfit2$BIC                    #Model is preferred
```

Note that covariate effects correspond to the order in which they were specified within the covar argument. It doesn't look here like gender affects propensity to send; given this, we might wonder whether dropping it gives us a better model.

```
classfit3<-rem.dyad(Class,n=20,effects=c("CovSnd","CovRec"),
    covar=list(CovSnd=cbind(ClassIntercept,ClassIsTeacher),
    CovRec=cbind(ClassIsTeacher,ClassIsFemale)),ordinal=FALSE,hessian=TRUE)
summary(classfit3)
classfit2$BIC-classfit3$BIC                    #Reduced model is indeed preferred
```

2.2 Endogenous social dynamics

The above model is still relatively poor, in the sense that the reduction in deviance is unimpressive. What else might explain classroom communication? Recency effects would seem to be a reasonable bet:

```
classfit4<-rem.dyad(Class,n=20,effects=c("CovSnd","CovRec","RRecSnd","RSndSnd"),
    covar=list(CovSnd=cbind(ClassIntercept,ClassIsTeacher),
    CovRec=cbind(ClassIsTeacher,ClassIsFemale)),ordinal=FALSE,hessian=TRUE)
summary(classfit4)
classfit3$BIC-classfit4$BIC                    #Enhanced model is preferred
```

This certainly helps, but we may suspect that more structure is present. Although a classroom is not as structured as a radio channel, we might reasonably expect to see at least modest adherence to conversational norms such as turn-taking. Moreover, sequential address and "hand-offs" might also be expected to occur more frequently here than would be expected by chance. To examine these possibilities, we incorporate the appropriate P-shift effects into our cumulative model:

```
classfit5<-rem.dyad(Class,n=20,effects=c("CovSnd","CovRec","RRecSnd","RSndSnd",
    "PSAB-BA","PSAB-AY","PSAB-BY"),covar=list(CovSnd=cbind(ClassIntercept,ClassIsTeacher),
    CovRec=cbind(ClassIsTeacher,ClassIsFemale)),ordinal=FALSE,hessian=TRUE)
summary(classfit5)
classfit4$BIC-classfit5$BIC                    #Enhanced model is again preferred
```

Note that, while P-shift effects are certainly present, including them has led the remaining gender effect to fall out. This suggests the possibility that what seemed at first to be a difference in communication receipt tendency by gender was in fact a result of social dynamics (perhaps stemming from the fact that the instructors are male, with their inherent tendency to communicate more often amplified by local conversational norms). Does dropping gender now result in improved model fit? Let's check.

```
classfit6<-rem.dyad(Class,n=20,effects=c("CovSnd","CovRec","RRecSnd","RSndSnd",
    "PSAB-BA","PSAB-AY","PSAB-BY"),covar=list(CovSnd=cbind(ClassIntercept,ClassIsTeacher),
    CovRec=ClassIsTeacher),ordinal=FALSE,hessian=TRUE)
summary(classfit6)
classfit5$AICC-classfit6$AICC                  #Reduced model is indeed preferred
```

At this point, we have a relatively simple model that incorporates some plausible social mechanisms. We could continue to elaborate it, but for instructional purposes we stop our search here.

2.3 Using a fitted model to investigate event timing

One use of a fitted relational event model is to consider the inter-event times predicted to be observed under various scenarios. For this purpose, it is useful to remember that, under the piecewise constant hazard assumption, event waiting times are conditionally exponentially distributed. This allows us to easily work out the consequences of various model effects for social dynamics, at least within the context of a particular scenario.

In interpreting coefficient effects, recall that they act as logged hazard multipliers. For instance:

```
exp(classfit6$coef["PSAB-BA"])  #Response events have apx 100 times the hazard of other events
```

Remember, however, that the fact that an event has an unusually high hazard does not mean that it will necessarily occur. For instance, while a response of B to a communication from A has a hazard that is (ceteris paribus) about 100 times as great as the hazard of a non B->A event, there are many more events of the latter type. Here, indeed, there are 379 other events

"competing" with the B->A response, and thus the chance that the latter will occur next is smaller than it may appear. Both relative rates and combinatorics (i.e., the number of possible ways that an event type may occur) govern the result.

One basic use of the model coefficients is to examine the expected inter-event times under specific scenarios. E.g.:

```
#Mean inter-event time if nothing else going on....
1/(20*19*exp(classfit6$coef["CovSnd.1"]))

#Mean teacher-student time (again, if nothing else happened)
1/(2*18*exp(sum(classfit6$coef[c("CovSnd.1","CovSnd.2")])))

#Sequential address by teacher w/out prior interaction, given a prior teacher-student
#interaction, and assuming nothing else happened
1/(17*exp(sum(classfit6$coef[c("CovSnd.1","CovSnd.2","PSAB-AY")])))

#Teacher responding to a specific student, given an immediate event
1/(exp(sum(classfit6$coef[c("CovSnd.1","CovSnd.2","PSAB-BA","RRecSnd")])))

#Student responding to a specific teacher, given an immediate event
1/(exp(sum(classfit6$coef[c("CovSnd.1","CovRec.1","PSAB-BA","RRecSnd")])))
```

Again, the number of ways that an event type can occur and the propensity of such events to occur both matter!

2.4 Assessing model adequacy

Model adequacy assessment in the exact timing case is much like that of the ordinal case. We cannot here use a fixed null residual or guessing equivalent, but can still look at "surprise" based on deviance residuals:

```
#Where is the model "surprised"?  Can't use null residual trick, but can see
#what the distribution looks like
hist(classfit6$residuals)                #Deviance residuals - lumpier by far, most smallish
```

The fit here doesn't seem to be as good as it was for the WTC police data. Let's look at classification:

```
mean(apply(classfit6$predicted.match,1,all))          #Exactly right about 33%
mean(apply(classfit6$predicted.match,1,any))          #Get one party exactly right 52%
colMeans(classfit6$predicted.match)                   #Better at sender than receiver!
classfit6$observed.rank
cbind(Class,c(classfit6$observed.rank,NA))
```

It looks like there is some structure in the errors: we aren't able to capture certain kinds of intrusive events. Does looking at the "surprising" events (say, those for which the observed event is not in the top 5% of those predicted) in time-aggregate form help?

```
#Get the surprising events, and display as a network
surprising<-as.sociomatrix.eventlist(Class[classfit6$observed.rank>19,],20)
gplot(surprising,vertex.col=4-2*ClassIsFemale,vertex.sides=3+ClassIsTeacher,vertex.cex=2)

#Show how the "surprising" events fit into the broader communication structure
edgecol<-matrix(rgb(surprising/(ClassNet+0.01),0,0),20,20)   #Color me surprised
gplot(ClassNet,edge.col=edgecol,edge.lwd=ClassNet^0.75,
    vertex.col=4-2*ClassIsFemale,vertex.sides=3+ClassIsTeacher,vertex.cex=2)
```

The visualization gives us more of a clue about what we're missing: various side discussions occur that are not well-captured by the current model. This could be due to the fact that things like P-shift effects fail to capture simultaneous side-conversations (each of which may have its own set of turn-taking patterns), or to a lack of covariates to capture the enhanced propensity of subgroup members to address each other. Further elaboration could be helpful here. On the other hand, we seem to be doing reasonably well at capturing the main line of discussion within the classroom, particularly vis a vis the instructors. Whether or not this is adequate depends on the purpose to which the model is to be put; as always, adequacy must be considered in light of specific scientific goals.

SECTION 3. GETTING STARTED WITH EGOCENTRIC RELATIONAL EVENT MODELS USING `rem` AND *informR*

While dyadic relational event models are of obvious interest to social network analysts, the *relevent* package can consider a much wider range of phenomena. The `rem` function within *relevent* provides a general tool that can be used to fit both dyadic and non-dyadic relational event models. `rem` also supports such exotica as multiple event types, dynamically varying support constraints, exogenous events, and fitting to multiple event sequences (as arise, e.g., from egocentric event reports). Here, we illustrate some basic applications of `rem` to egocentric relational event models. To facilitate creation of sufficient statistics for these types of models, we employ the *informR* library; *informR* is designed to facilitate modeling of event dynamics with complex dependencies (particularly involving multiple event types) with the *relevent* package.

3.0 Load the informR library

```
library(informR)            # Load the library (this also loads the relevent package)
```

3.1 Load, examine, and prepare the data

The example data are packaged with *informR*. They come from the American Time Use Survey (US Census), subset of individuals 80 years and older. The records are activity histories from a "day-in-a-life" of the respondents. We have included two versions of the same data: an event order-only and spell-interval data.

```
data(atus80ord,atus80int)  # Load the data.

atus80ord[1:5,]            # Ordinal data are just a list of activities

atus80int[1:10,]           # Interval data are pairs of 'starting' and 'stopping' events
                           # Naturally, nrow(atus80ord)*2==nrow(atus80int).
```

The *relevent* package fitting routines do not currently handle missing data. However, we can treat NAs as their own event type and retain their contribution to the likelihood of the event history.

```
atus80ord[which(is.na(atus80ord[,"Activities"]))
   ,"Activities"]<-"MISSING"
rawevents<-cbind(atus80ord$Activities, atus80ord$TUCASEID)  # This is our data
```

3.2 Constructing eventlists and statslists

```
?rem   # recall that rem requires (at minimum) two objects: eventlist and statslist
```

One important feature of general relational event models is the ability to incorporate exogenous events into the event history. Exogenous events are outside the set of events that a focal actor controls directly but might influence the set of actions that actor does next. The rem() function uses a special token ("0") in the eventlist to indicate which events are exogenous in the system. We do not have any truly exogenous events in the example data. For illustrative purposes only, we'll treat missing events as exogenous here by passing the event type "MISSING" to the null.events argument in the gen.evl() function that creates eventlists.

```
evls<-gen.evl(rawevents,null.events=c("MISSING"))     # The gen.evl() function stores the
                                                      # eventlist and some meta-data in a
                                                      # list.
names(evls)   # See the structure of the evls object
length(evls$eventlist) # N

evls$eventlist[[1]]        # The first event history in the eventlist
evls$event.key      # The event.key maps the tokens to the event type names.

evls$null.events    # We stored activity types ``MISSING'' as exogenous as an example only.

# Generate a statslist from evls with the baseline (reference)
# event type set to 'Sleeping .' gen.intercepts() is a special
# function specifically for creating baseline intercept models:
alpha.ints<-gen.intercepts(evls,basecat="Sleeping",type=1)

# Examine the structure of the alpha.ints statslist
length(alpha.ints)  # N
```

```
dim(alpha.ints[[1]][[1]])  # The dimensions of the array corresponding to the the first
                           # eventlist in the global position of the statslist should be
                           # seven events, thirteen event types, and twelve sufficient
                           # statistics.

alpha.ints[[1]][[1]][1,,]  # The contrast matrix for the first event in the global position
                           # of the statslist .
```

3.3 Fitting baseline models

```
# Fit and summarize a simple baseline model, with weakly informative t-dist priors
alpha.fit<-rem(eventlist=evls$eventlist,statslist=alpha.ints,
   estimator="BPM",prior.param=list(mu=0,sigma=100,nu=4))

summary(alpha.fit) # Summarize the model
```

The coefficients (relative hazards) are proportional to the relative rates of occurrence in the data. We can see that Travel occurs about as often as Sleeping and that Working is nearly 4 times less likely than Sleeping in this population. In this simple baseline model without any relational event dynamics, the coefficients happen to be the MLEs under an inhomogeneous Poisson, which we can check numerically:

```
pois.mle<-log(prop.table(table(atus80ord$Activities))[-c(7,10)]
   /prop.table(table(atus80ord$Activities))[10])
round(cbind(BPM=alpha.fit$coef[order(names(alpha.fit$coef))],pois.mle),4)
```

3.4 Creating sequence (s-form) sufficient statistics

While baseline models are important components of research, most users will be interested in modeling the relational event dynamics. The sufficient statistics to model event sequences are generated by two functions: `gen.sformlist()`, for basic or general sequences, and `glb.sformlist()` for advanced or global sequences. The *informR* tools employ a simple alphabetic token system to represent sequence terms. For example, if event type ``Sleeping" has token "a" and event type "eating" has token "b", then the Sleeping→Eating sequence term would be specified as "ab."

We'll fit a model with baseline hazards (as above in alpha.fit) plus terms for simple digrams of inertial effects (i.e, a→a)

```
a1<-paste(evls$event.key[-9,1],evls$event.key[-9,1],sep="")        # All inertial terms
a1   # See what the inertial s-forms look like as tokens.

#These models can take a long time to run so we're gonna subset the data first
set.seed(570819)
inds<-sample(1:length(evls$eventlist),500)
evls$eventlist<-evls$eventlist[inds]
alpha.ints<-alpha.ints[inds]


beta.sforms<-gen.sformlist(evls,a1)      # Use the gen.sformlist() function to generate basic
                                         # transition statistics . See ?gen.sformlist.

# Examine the structure of the beta.sforms object and it's relationship to evls$eventlist
evls$eventlist$"20031009033366"  # this actors eventlist
sapply(attr(evls$eventlist$"20031009033366","char")[1:3],sf2nms,event.key=evls$event.key)

beta.sforms$"20031009033366"[1:4,,c("aa","bb")]       # a happens at t1, so "a" at t2 gets a
                                                      # 1. Then, b happens at t2 and t3, so
                                                      # "b" at t3 and t4 get a 1 .

# Now, we want to incorporate these inertial statistics into a model with the baserates .
# Append the inertial sformlist to the baserate statslist using the slbind function,
# the "type" argument governs whether these are "global" (1) or "local" (2) effects in the
# statslist.

beta.ints<-slbind(beta.sforms,alpha.ints,type=1,new.names=TRUE,
   event.key=evls$event.key)
```

```
# Fit the model
beta.fit<-rem(evls$eventlist,beta.ints,estimator="BPM",
   prior.param=list(mu=0,sigma=100,nu=4))
```

A significant positive coefficient on the inertial statistics indicates that, given that event type "a" just occurred, the next event is more likely to also be "a." By extension, a negative coefficient means that inertia is contraindicated for that event type. Here, we see mainly negative coefficients suggesting that most older people go about their daily lives in task-switching routines rather than doing a series of the same types of activities. The obvious exceptions here as caregiving, volunteering, and working.

```
round(cbind(BPM=beta.fit$coef[13:25],Z=beta.fit$coef[13:25]/beta.fit$sd[13:25]),4)

# Note that we could have used the sfl2statslist() function to construct an inertial model
# without the baserates but it wouldn't make much sense:
beta.sansints<-sfl2statslist(beta.sforms) #Just an example of how to do this.
```

Now let's consider slightly more complex dynamics. In particular, we'll model two sequences believed to be involved in routine morning behavior that lead to having breakfast: 1) Sleeping→Personal Care OR Priv.Personal Care→Eating, and 2) Sleeping →Sequence of Housework→Eating. The two types of personal care activities are an artifact of the data coding on the ATUS; we don't care which specific type of personal care activities the actors do, so we can model them with a divergence "OR" term. Also, the "sequence of housework" term attempts to capture the effect that, say picking up yesterday's clothes from the floor, cleaning up last nights dishes, and preparing breakfast might precede eating breakfast in unknown duration or order: we incorporate this type of statistic with a regex "AND" operator.

```
a2<-c("a(c|h)b","ad+b")
gamma.sforms<-gen.sformlist(evls,a2)
gamma.ints<-slbind(gamma.sforms,beta.ints,new.names=TRUE,event.key=evls$event.key)

# Fit and summarize the model
gamma.fit<-rem(evls$eventlist,gamma.ints,estimator="BPM",
   prior.param=list(mu=0,sigma=100,nu=4))
summary(gamma.fit) # Both of the terms are significantly positive!
```

Suppose we want a global effect for the overall tendency for having sleep interrupted by another activity. We can us the `glb.formlist()` function to pool multiple s-forms into a single global sufficient statistic that captures the overall effect we are interested in modeling. Here, we'll create an s-form for: Sleep→Some Other Activity→Sleep:

```
sleep.sfs<-paste("a",letters[2:14],"a",sep="")
sleep.sfs

# Use glb.sformlist() to create this statistic .
# This function pools multiple s-forms into single statistics.
sleep.sforms<-glb.sformlist(evls,sforms=list(sleep.sfs),new.names="InterSleep")
sleep.ints<-slbind(sleep.sforms, gamma.ints)
sleep.fit<-rem(evls$eventlist,sleep.ints,estimator="BPM",
   prior.param=list(mu=0,sigma=100,nu=4))
round(cbind(BPM=sleep.fit$coef,Z=sleep.fit$coef/sleep.fit$sd)[26:28,],4)
```

3.5 Modifying statslists

```
# statistics can be dropped from statslists using the sldrop() function
delta.ints<-sldrop(sleep.ints,varname=c("InterSleep","EducationEducation"))
delta.fit<-rem(evls$eventlist,delta.ints)      # fit it.
names(delta.fit$coef)      # See that coefs were dropped

c(delta.fit$BIC,sleep.fit$BIC)    # Comparing models, the more parsimonious one wins!
```

We can also incorporate some limited covariates using *informR*. Here we'll interact an indicator for events done by women with the eating statistics contained in `gamma.stats`. We'll hypothesize that women and men are equally likely to conduct personal care after waking but before eating breakfast and that, do to gender norms, women are more likely to do some housework before eating breakfast.

```
sl.ind<-26:27 # The eating statistics in gamma.ints are located in columns 26 through 27
# Which event histories are done by women?
fem.evs<-unlist(glapply(atus80ord$SEX,atus80ord$TUCASEID,unique,regroup=FALSE))
```

```
fem.evs<-ifelse(fem.evs==2,1,0)[inds]

# Create the new statslist using slbind.cond ()
# then fit and examine:
gamma.ints2<-slbind.cond(fem.evs,gamma.ints,sl.ind=sl.ind,var.suffix="FEM")

sl.ind<-26:29# The eating statistics in gamma.ints2 are located in columns 26 through 29
gamma.fit2<-rem(evls$eventlist,gamma.ints2,estimator="BPM",
   prior.param= list(mu=0,sigma=100,nu=4))
round(cbind(BPM=gamma.fit2$coef[sl.ind],Z=gamma.fit2$coef[sl.ind]/
   gamma.fit2$sd[sl.ind]),4)      # It appears that our hypotheses are supported!
```

SECTION 4. ADVANCED RELATIONAL EVENT MODELS WITH `rem` AND *informR*

Having had a taste of what can be done with `rem` and *informR*, we now consider some more advanced models. In particular, we here introduce modeling of spell (i.e., time interval) data using *relevent*. Spell data provides a simple illustration of the use of support constraints, and is a common type of data encountered in a life history context.

4.0 Getting Started with Spell-Interval Data

In general, interval and ordinal data are handled the same by the *informR* tools with a couple of important differences to the main `gen.evl()` and `gen.intercepts()` functions. In particular, the `eventlist` argument of `gen.evl()` now gets a three column matrix where the first column indexes the event type, the second the time that the event occurred, and the third is an actor unique identifier. Also, we do not need to use "relative events" in the interval timing likelihood model: therefore, setting the `contr=FALSE` argument in `gen.intercepts()` ensures that the appropriate identity matrix is constructed in the baseline model statslist.

```
evlsint<-gen.evl(atus80int[,1:3])       # Interval data evls object

# These models tend to use a lot of memory, so we'll only use a subset
# of the data
set.seed(570819)
evlsint$eventlist<-evlsint$eventlist[sample(1:length(evlsint$eventlist),500)]

evlsint$event.key   # Examine the tokens.

evlsint$eventlist$"20040706041558"      # Examine an eventlist structure
                                        # This person wakes up, has a bite to eat (which
                                        # takes about 30 minutes), and then does some
                                        # cleaning up.

# Create intercepts-only model, must set contr=FALSE
int.base<-gen.intercepts(evlsint, type=1, contr=FALSE)
```

A careful observer will note that not all events are possible at all times. In particular, any Starting event of a particular activity must be followed by a Stopping event of the same activity. The `rem` function has an argument "`supplist`" in which a list of dynamic support constraints can be passed. We'll define such a support list for this constraint.

```
# Generate a supplist object for this data.
# Briefly, for each actor, this list contains an Event-by-Event-Type matrix where a 1
# indicates that the current event is possible at that time and 0 indicates otherwise.
eT<-evlsint$event.key[,2]
supplist<-list()
for(i in 1:length(evlsint$eventlist)){
   supplist[[i]]<-matrix(0,nrow=NROW(evlsint$eventlist[[i]]),ncol=length(eT))
   supplist[[i]]
   colnames(supplist[[i]])<-eT
   }
for(i in 1:length(evlsint$eventlist)){
   evl<-evlsint$eventlist[[i]]
   supplist[[i]][1,grep("START",eT)]<-1
   for(j in 2:(NROW(evl))){
      CE<-evl[j,1]
      PE<-evl[j-1,1]
      if(PE==0){supplist[[i]][j,grep("STOP",eT)]<-1}
      if(PE!=0){
         if(grepl("START",eT[PE])){supplist[[i]][j,eT[CE]]<-1}
         if(grepl("STOP",eT[PE])){supplist[[i]][j,grepl("START",eT)]<-1}
      }
   }
}
```

4.1 Building and Fitting Relational Models Using Spell-Interval Data

Interval timing likelihood models are fit using the `rem` function, with arguments `timing="interval"` set and, as in this case, any dynamic support constraint list passed to argument `supplist`.

```
# Fit and summarize the baserate model.
intfit.base<-rem(evlsint$eventlist,int.base,supplist=supplist,
   timing="interval",estimator="BPM")   # Note the new arguments.
summary(intfit.base)
```

With spell-interval data we have two sets of coefficients (-log hazards): coefficients for starting and stopping effects that correspond to effects for initiating a spell and the duration of a spell, respectively. The -log hazards of the starting events are proportional to the prevalence in the data (the probability of the event occurring). The -log hazards of the stopping events are proportional to the average spell durations. Both of these can be numerically checked:

```
# The -log hazards of the starting events are proportional to the prevalence
bpm.start<-(exp(intfit.base$coef[grep("START",names(intfit.base$coef))])
   /sum(exp(intfit.base$coef[grep("START",names(intfit.base$coef))])))
in.ids<-which(atus80int$TUCASEID%in%names(evlsint$eventlist)
   & grepl("START",atus80int$Events))
mle.start<-prop.table(table(atus80int[in.ids,"Events"]))

round(cbind(BPM=sort(bpm.start),MLE=sort(mle.start)),4) # Check the starting event results.

# the -log hazards of the stopping events are proportional to the average spell durations
(1/exp(-5.690678))/60 # Check how long does a typical sleep spell last (in hours)?

# Let's fit some models with the interval data
# Sleep->Any Spell->Start Sleeping
sleepA<-paste("ab",c(letters[seq(1,26,2)],"A"),
   c(letters[seq(2,26,2)],"B"),"a",sep="")
#Sleep->Any Spell->Duration of sleep
sleepB<-paste("ab",c(letters[seq(1,26,2)],"A"),
   c(letters[seq(2,26,2)],"B"),"ab",sep="")

#Use glb.sformlist() to construct the above
# plus Sleep->PerCare-Sleep statistics.
# The cond=TRUE makes these "conditional" statistics
# and do not affect the hazard of the prefix events .
# warning: could take a long time (~2 mins on 2.25GHz processor)
# There is a short-cut below, uncomment the following to skip the short-cut
#sleep.glbs<-glb.sformlist(evlsint,
#   list(sleepA,sleepB,c("abefa","abopa"),c("abefab","abopab"))
#   ,cond=TRUE,new.names=c("Inter.Sl.Ons","Inter.Sl.Dur"
#   ,"PerCare.Sl.Ons","PerCare.Sl.Dur"))
load("rem.int.Rdata") #The short-cut

sleep.int<-slbind(sleep.glbs,int.base)  # Concatenate all the statistics

# Fit and summarize the model
intfit.sleep<-rem(evlsint$eventlist,sleep.int,supplist=supplist,
   timing="interval",estimator="BPM")
summary(intfit.sleep)
```

The global sleep interruption coefficient is, as in the ordinal case, negative – though the effect on the duration of sleep following an interruption is positive (it multiples the duration by exp(0.35), roughly 2 hours of additional sleep). When sleep was followed by personal care, the odds of going back to sleep are high for this population, as indicated by the positive coefficient for PerCare.Sl.Ons, but note that this particular type interruption does not disturb the duration of sleep.

Finally, many researchers will want to express the initiation effects from spell-interval data as probability statements. This is accomplished using some basic algebra and the interval timing likelihood model:

```
#Conditional probability of going back to sleep after having awoken to do something else
exp(-0.890707+4.989059)/
   sum(exp(intfit.sleep$coef[grep("START",names(intfit.sleep$coef))]))

#Given sleep->personal care, how often does personal care->sleep?
exp(1.314628+4.989059)/
    sum(exp(intfit.sleep$coef[grep("START",names(intfit.sleep$coef))]))
```