



Constructing and Modifying Sequence Statistics for relevent Using informR in R

Christopher Steven Marcum
National Institutes of Health

Carter T. Butts
University of California, Irvine

Abstract

The **informR** package greatly simplifies the analysis of complex event histories in R by providing user friendly tools to build sufficient statistics for the **relevent** package. Historically, building sufficient statistics to model event sequences (of the form $a \rightarrow b$) using the egocentric generalization of Butts' (2008) *relational event framework* for modeling social action has been cumbersome. The **informR** package simplifies the construction of the complex list of arrays needed by the `rem()` model fitting for a variety of cases involving egocentric event data, multiple event types, and/or support constraints. This paper introduces these tools using examples from real data extracted from the American Time Use Survey.

Keywords: relational events, sequence statistics, **relevent**.

1. Introduction

The relational event framework (Butts 2008) is an approach to modeling complex interaction sequences (e.g., among social actors, or between actors and their environments) as discrete events in continuous time. The **relevent** package (Butts 2015) for R (R Core Team 2014) provides one implementation of this framework, with specialized functionality for easily fitting dyadic relational event models, as well as a general purpose function, `rem()`, that supports a broader class of models (including models with multiple event types, endogenous support constraints, etc.). Unfortunately, `rem()` requires the use of user-generated inputs that can be quite difficult to construct, historically limiting its usefulness. The **informR** package (Marcum 2015) for R contains a set of functions that assist in the creation and manipulation of sufficient statistics to be modeled using the `rem()` function, thereby broadening the class of models that can be easily fit using **relevent**. In the simplest case, and given some data, the package creates the two objects required by the `rem()` regression function for ego-centric relational event model

fitting: `eventlist` and `statslist`. These two objects can be complex and non-intuitive to create without specialized code; **informR** greatly simplifies the procedure. The package allows users to specify sufficient statistics for general families of event sequences (or patterns, called *s-forms* as introduced below) to include in `statslists`. Additionally, tools for adding and dropping covariates from `statslists` are included in the package. This paper illustrates how to use the **informR** package using included data from a subset of the US Department of Labor Statistics’ *American Time Use Survey*. Package **informR** is available from the Comprehensive R Archive Network (CRAN) at <http://CRAN.R-project.org/package=informR>.

While **informR** is intended to interface with **relevent**, we note that various other packages for R support event modeling through survival analysis (often called event history analysis in the social sciences, or failure/reliability analysis in engineering fields; Blossfeld and Rohwer 2002). Mills (2011) provides a good introduction to those packages and methods involving the estimation of event hazards for, e.g., the timing of death or the likelihood of marrying by a particular age. Additionally, in the CRAN Task View on “Survival Analysis” (Allignol and Latouche 2014) a list is maintained that describes packages for the analysis of time to event data. Packages such as **survival** (Therneau and Grambsch 2000; Therneau 2014) and **muhaaz** (Gentleman 2010) can be used to fit closely related models, although they do not currently support the full range of models described e.g., by Butts (2008). Users of these packages may be able to employ **informR** to help generate inputs for their associated functions as well, although we do not explore these potential applications in this paper.

1.1. The relational event modeling framework

Before discussing the use of the **informR** package, it is useful to briefly review the relational event modeling framework. A *relational event* as discussed by Butts (2008) is a discrete event (taken to have zero duration in continuous time) in which some actor (e.g., person, organization, or other entity) directs some action towards some target (e.g., another person, a location or object, or even the actor him/her/itself).¹ Events may be of distinct types, but are here treated as otherwise unvalued (though see e.g., Brandes, Lerner, and Snijders (2009) for valued extensions); the risk set of events at any given time point (a subset of the Cartesian product of the set of possible senders, possible receivers, and possible event types) may be fixed, or may itself evolve as a function of the past event history. These features make the relational event framework useful for modeling complex social dynamics (e.g., conversation, radio or email communication, or adversarial interactions) or behavioral sequences involving complex interactions between an individual and his or her environment (so-called “egocentric” relational events).

Most relational event modeling to date has focused on the case in which event hazards can be taken to be piecewise constant functions of sender, receiver, or higher-order covariates, and the past event history. The inputs to these functions are taken to represent factors that enhance or inhibit the propensity for actors to take particular actions, as determined by their associated parameters. A typical development (per Butts 2008, pp. 159–166) is as follows. Let S be the set of all potential senders, R the set of all potential receivers, and C the set of all potential event types. A relational event a is defined as a tuple $(s, r, c, t) \in S \times R \times C \times \mathbb{R}$, with

¹By considering sets of individuals or objects to be valid senders or receivers one can likewise represent “hypergraphic” events involving multiple interactants; likewise, actions emitted by a sender having no well-defined target can be represented by adding a “null receiver” to the set. For an example of the former, see e.g., DuBois, Butts, McFarland, and Smyth (2013).

the last element representing the time at which the event occurs. For notational convenience, we also define $s(a) \in S$, $r(a) \in R$, $c(a) \in C$, and $\tau(a) \in \mathbb{R}$ to be the event's sender, receiver, type, and time (respectively). Let $A_t = (a_1, \dots, a_M)$ be the set of all events to have occurred by time t (i.e., the *event history* at t); the set of possible (i.e., non-zero hazard) events at t is then designated by $\mathbb{A}(A_t) \subseteq S \times R \times C \times \{t\}$. The hazard of an arbitrary event, a , at time t is then parameterized as

$$\lambda(a, \theta, X_t) = \begin{cases} \exp[\theta^\top u(s(a), r(a), c(a), A_t, X_t)] & a \in \mathbb{A}(A_t) \\ 0 & a \notin \mathbb{A}(A_t) \end{cases} \quad (1)$$

$$\equiv \lambda_{aA_t\theta}, \quad (2)$$

where u is a p -vector of statistics, $\theta \in \mathbb{R}^p$, and X_t is a set of covariates. In the piecewise constant model, it is assumed that hazards change only when events occur (hence λ does not depend directly on t), and hence X_t may likewise change only at realized events. The impact of this latter restriction may be easily relaxed by the introduction of *exogenous events* (not discussed in Butts 2008), as described below. We note that while the log-linear form is a natural choice for hazard modeling (as it guarantees non-negativity and allows coefficients to be interpreted in terms of logged rate multipliers), other functions can also be employed; since the **relevent** package supports only the log-linear case, this is our focus here.

Given the above formulation, the development of the likelihood for an observed event history is fairly straightforward. We extend the above-cited development only by the introduction of exogenous events, which are implemented in package **relevent** but were not discussed in the above paper. We assume that we observe an event history A_t over time interval $[0, t)$, treating time 0 as the onset of risk for all potential events; for notational convenience, we define the null event a_0 with $\tau(a_0) = 0$ to mark the onset of the observation period. In general, we model the event history as being the result of a continuous time event process in which the hazard of any potential event is given by Equation 1. Some events, however, may be considered exogenous, in the sense that (1) their hazards are assumed unrelated to the observed events, and (2) their hazards are not a function of θ . Exogenous events may represent the actions of environmental or other entities that affect (but are not affected by) the system under study, or they may be used as “book keeping” devices to model systems in which hazards change for reasons other than endogenous events. For instance, “clock events” that occur deterministically at fixed periods in time (e.g., every hour, day, etc.) can be used to model changes in hazards due to time period, and exogenous events can also be used to capture the impact of changes in covariates (i.e., X_t changing at times other than those associated with endogenous events). We here denote endogeneity via the indicator ϵ , such that $\epsilon(a_i) = 1$ if $a_i \in A_t$ is endogenous, and 0 otherwise. Subject to this minor extension, the joint likelihood of A_t under the above-described process becomes (per *ibid.*, Equation 2)

$$p(A_t | \theta, X) = \prod_{i=1}^M \left[\left(\lambda_{a_i A_{\tau(a_{i-1})} \theta} \right)^{\epsilon(a_i)} \prod_{a' \in \mathbb{A}(A_{\tau(a_i)})} \exp \left[-\lambda_{a' A_{\tau(a_{i-1})} \theta} (\tau(a_i) - \tau(a_{i-1})) \right] \right] \\ \times \prod_{a' \in \mathbb{A}(A_\tau)} \exp \left[-\lambda_{a' A_\tau \theta} (t - \tau(a_M)) \right]. \quad (3)$$

Butts (2008) also considers the case in which the temporal order of events is known, but the event times are not. The observed data likelihood of A_t here becomes a product of categorical

distributions with outcome probabilities proportional to the event hazards. With the minor addition of exogenous events, this likelihood is given by (per *ibid.*, Equation 3)

$$p(A_t|\theta, X) = \prod_{i=1}^M \left[\frac{\lambda_{a_i A_{\tau(a_{i-1})}} \theta}{\sum_{a' \in \mathbb{A}(A_{\tau(a_i)})} \lambda_{a' A_{\tau(a_{i-1})}} \theta} \right]^{\epsilon(a_i)}, \quad (4)$$

which depends only on the hazards of the potential and realized events (and not the inter-event times). Note that, like Equation 3, the likelihood of Equation 4 depends upon the assumption of piecewise constant hazards.

The estimation function `rem()` in the **relevent** package, along with its dyadic-data cousin `rem.dyad()`, fits both ordinal and interval time relational event models (Butts 2015). As discussed below, the interval time likelihood model is fit by supplying data on event types and complete timing of events to `rem()` with timing argument `timing = "interval"`, while the ordinal time model is fit by supplying only the ordered events with `timing = "ordinal"`. The scope of the discussion here will cover how to construct model statistics and to format data in both cases using the **informR** toolkit.

As discussed in detail in subsequent sections, the basic model specification for `rem()` involves two arguments: the `eventlist` (a matrix, or list thereof, containing the observed event sequence in one column and, optionally, the timing information in another) and the `statslist` (a three dimensional – event number by event type by statistic – array, or list thereof, containing the sufficient statistics for the model). Package **informR** aids in the construction of these objects. The complete signature of `rem()` is documented as follows:

```
rem(eventlist, statslist, supplist = NULL, timing = c("ordinal",
  "interval"), estimator = c("BPM", "MLE", "BMCMC", "BSIR"),
  prior.param = list(mu = 0, sigma = 1000, nu = 4), mcmc.draws = 1500,
  mcmc.thin = 25, mcmc.burn = 2000, mcmc.chains = 3, mcmc.sd = 0.05,
  mcmc.ind.int = 50, mcmc.ind.sd = 10, sir.draws = 1000, sir.expand = 10,
  sir.nu = 4, verbose = FALSE).
```

As there are no “canned” sufficient statistics for the `rem()` model-fitting routine, users must supply their own properly formatted data structure to `eventlist` and array of sufficient statistics to `statslist`. The purpose of package **informR** is to simplify the construction of the `eventlist` and `statslist` objects for a variety of egocentric relational event models, greatly reducing the difficulty of specifying and fitting such models in practice. The balance of the paper focuses on the **informR** tools, and how they may be used to assist with common modeling tasks of sequence analysis in the REM framework.

2. Getting started

The **informR** package can be downloaded and installed from CRAN; the package depends upon the **abind** (Plate and Heiberger 2011) and **relevent** packages for R, which can also be obtained from CRAN as of the time of this writing. Load the package in the usual way using `library("informR")`. The example data used in this paper is packaged in **informR** and can be loaded with `data("atus80ord", package = "informR")` and `data("atus80int", package = "informR")`, respectively for the ordinal and interval cases. The only individual level covariate included in this subset is the sex of the respondent (1 = *Male*).

The example data come from the pooled 2003–2008 American Time Use Survey (ATUS). The ATUS is a nationally representative sample of the non-institutionalized adult US population. The survey collects information on how people spend their time during a single randomly selected day. The type of each activity and its duration is recorded. The included subset represents event histories from respondents aged 80 and above. In addition to the activity variables, unique identifiers and respondent’s gender are also included in the subset. The two `data.frames` that hold the event history data differ in that `atus80ord` contains only sequences of activity spells and `atus80int` restructures each activity spell observation into two event types, a starting event and a stopping event. Thus, in `atus80ord`, the first informant’s (20030101031049) first activity spell is coded as `Sleeping` but in `atus80int`, this same activity spell is recorded as two observations `Sleeping|START` and `Sleeping|STOP` and associated with event timing in the `Time` column. Naturally, `nrow(atus80int) == 2 * nrow(atus80ord)`. There are 62,352 and 124,704 respective observations on 3,430 individuals. To make this clear, consider the print output of the first five activity spells from the following code snippet:

```
R> library("informR")
R> data("atus80ord", package = "informR")
R> data("atus80int", package = "informR")
R> atus80ord[1:5, ]
```

	Activities	TUCASEID	SEX
1348	Sleeping	20030101031049	2
1349	Eating	20030101031049	2
1350	Private personal care	20030101031049	2
1351	Household Production	20030101031049	2
1352	Eating	20030101031049	2

```
R> atus80int[1:10, ]
```

	Events	Time	TUCASEID	SEX
200301010310491	Sleeping START	0	20030101031049	2
200301010310492	Sleeping STOP	240	20030101031049	2
200301010310493	Eating START	240.001	20030101031049	2
200301010310494	Eating STOP	270	20030101031049	2
200301010310495	Private personal care START	270.001	20030101031049	2
200301010310496	Private personal care STOP	300	20030101031049	2
200301010310497	Household Production START	300.001	20030101031049	2
200301010310498	Household Production STOP	780	20030101031049	2
200301010310499	Eating START	780.001	20030101031049	2
2003010103104910	Eating STOP	810	20030101031049	2

The `Time` column of the `atus80int` data set records the time, in cumulative minutes, that the event took place in each respondent’s event history. The duration elapsed between each pair of “starting” and “stopping” events constitute a spell of that type of activity. Each “stopping” event is offset from the next starting event by 0.001 minutes, which enforces the “no simultaneous events” assumption of the relational event framework. For example, actor

20030101031049 started sleeping at time 0, woke up 4 hours later (240/60), then she began to eat breakfast a fraction of a second later, which took about 30 minutes to eat etc.

Having summarized the structure of the example data, we next discuss how package **informR** can be used to construct sequence statistics for the `rem()` function. The first step is to create an `eventlist` object using the `gen.ev1()` function. For ordinal time relational event models, the `gen.ev1()` function takes two arguments: (1) `eventlist`, which is a two-column matrix or a data frame consisting of event observations in the first column and an event history grouping factor in the other (e.g., a unique informant id), and (2) an optional parameter called `null.events` which is a character vector of event types that should be treated as exogenous or otherwise not modeled directly. In this case, we are going to use the respondent id variable, `TUCASEID`, as the grouping factor and since there are no truly exogenous events in this data, we treat missing spell types (NA) as exogenous (i.e., for illustrative purposes only):

```
R> atus80ord[which(is.na(atus80ord[, "Activities"])), "Activities"] <-
+   "MISSING"
R> rawevents <- cbind(atus80ord$Activities, atus80ord$TUCASEID)
R> evls <- gen.ev1(rawevents, null.events = "MISSING")
R> names(evls)
```

```
[1] "eventlist"   "event.key"    "null.events"
```

```
R> evls$eventlist[[1]]
```

```
[1] 1 2 3 4 2 4 1
attr(,"char")
[1] "a" "b" "c" "d" "b" "d" "a"
```

```
R> evls$event.key
```

```
      id event.type
[1,] "a" "Sleeping"
[2,] "b" "Eating"
[3,] "c" "Private personal care"
[4,] "d" "Household Production"
[5,] "e" "Travel"
[6,] "f" "Communication"
[7,] "g" "Leisure"
[8,] "h" "Personal Care"
[9,] "i" "MISSING"
[10,] "j" "Waiting"
[11,] "k" "Volunteering"
[12,] "l" "Caregiving"
[13,] "m" "Education"
[14,] "n" "Work Production"
```

```
R> evls$null.events
```

```
[1] "MISSING"
```

The resulting object `evls` stores the eventlist in `evls$eventlist`. In `evls$eventlist` each element is a vector of numeric unique event type ids, which have a corresponding alphabetic attribute that is used in regular expression matching in other methods – the character vector attribute of the first eventlist can be obtained using `attr(evls$eventlist[[1]], "char")`. The package currently supports event histories with 52 event types or less. The event types are mapped to their respective alphabetic ids in `evls$event.key`. Because we passed a vector to `null.events` in `gen.ev1()`, the `evls` object contains a list of the exogenous events in `evls$null.events`. Correspondingly, any event type listed in `null.events` is assigned a numeric id value of 0 in the eventlist for consistency with how `rem()` handles exogenous events.²

Because event histories with fewer than two events cannot identify standard relational event models, any event history in the data having fewer than two events is dropped and a warning is issued. Additionally, `gen.ev1()` can be used to build eventlist objects for the interval time relational event model; the only difference is that `eventlist` is passed as a three-column matrix where the first two columns index the events and the temporal information, respectively, and the third column indexes the event history grouping factor. Finally, as `gen.ev1()` conditions on the observed event types to generate the event key, users may need to append non-observed yet potential events to it after the fact. We revisit both of these latter points below.

With the `eventlist` stored in `evls$eventlist`, we can create the second object required to fit a model using `rem()`, the `statslist`. A simple `statslist` object will store the intercept sufficient statistics for the egocentric relational event model. This is accomplished through the `gen.intercepts()` function. This function takes four arguments, `ev1`, `basecat`, `type` and `contr`. The `ev1` argument expects an object passed from `gen.ev1()`, as above. The `basecat` parameter allows a user to specify which event type should be treated as the baseline category for the ordinal relational event model – if not specified, the default behavior is to take the first event type in alphabetic order. The `type` argument is used to indicate whether effects specified within the `statslist` are to be considered “global” (`type = 1`) or “local” (`type = 2`) by `rem()`; global effects are assumed to be homogeneous (i.e., pooled) across event histories, while local effects are allowed to vary for each event history. The default behavior is to assign all statistics to be global. Though not used in this example, the logical argument `contr` is used to indicate whether or not the resulting `statslist` should include contrasts by dropping a baseline category. When generating statistics for the interval relational event model, this parameter should usually be passed as `contr = FALSE` – which also overrides any value passed to `basecat`.

```
R> alpha.ints <- gen.intercepts(evls, basecat = "Sleeping")
R> length(alpha.ints)
```

```
[1] 3430
```

```
R> dim(alpha.ints[[1]][[1]])
```

```
[1] 7 13 12
```

²Exogenous events are not directly modeled but may affect the sufficient statistics of an event sequence, as described in Section 1.1. See the documentation in `help("rem")` for additional details.

```
R> alpha.ints[[1]][[1]][1, , ]
```

	Eating	Private personal care	Household Production	Travel
Sleeping	0	0	0	0
Eating	1	0	0	0
Private personal care	0	1	0	0
Household Production	0	0	1	0
Travel	0	0	0	1
Communication	0	0	0	0
Leisure	0	0	0	0
Personal Care	0	0	0	0
Waiting	0	0	0	0
Volunteering	0	0	0	0
Caregiving	0	0	0	0
Education	0	0	0	0
Work Production	0	0	0	0

	Communication	Leisure	Personal Care	Waiting	Volunteering
Sleeping	0	0	0	0	0
Eating	0	0	0	0	0
Private personal care	0	0	0	0	0
Household Production	0	0	0	0	0
Travel	0	0	0	0	0
Communication	1	0	0	0	0
Leisure	0	1	0	0	0
Personal Care	0	0	1	0	0
Waiting	0	0	0	1	0
Volunteering	0	0	0	0	1
Caregiving	0	0	0	0	0
Education	0	0	0	0	0
Work Production	0	0	0	0	0

	Caregiving	Education	Work Production
Sleeping	0	0	0
Eating	0	0	0
Private personal care	0	0	0
Household Production	0	0	0
Travel	0	0	0
Communication	0	0	0
Leisure	0	0	0
Personal Care	0	0	0
Waiting	0	0	0
Volunteering	0	0	0
Caregiving	1	0	0
Education	0	1	0
Work Production	0	0	1

Once both an `eventlist` and a `statslist` is generated, we can fit a baseline egocentric discrete relational event model using `rem()`. In this example, we employ the Bayesian posterior

mode (BPM) estimator, along with a weakly informative independent Student's t prior:

```
R> alpha.fit <- rem(eventlist = evls$eventlist, statslist = alpha.ints,
+   estimator = "BPM", prior.param = list(mu = 0, sigma = 100 , nu = 4))
R> summary(alpha.fit)
```

Egocentric Relational Event Model (Ordinal Likelihood)

	Post.Mode	Post.SD	Z value	Pr(> z)	
Eating	-0.0524425	0.0156947	-3.3414	0.0008335	***
Private personal care	-0.6843110	0.0189127	-36.1827	< 2.2e-16	***
Household Production	0.3306130	0.0143563	23.0292	< 2.2e-16	***
Travel	0.0078847	0.0154572	0.5101	0.6099800	
Communication	-0.7191476	0.0191342	-37.5844	< 2.2e-16	***
Leisure	0.5687237	0.0137056	41.4956	< 2.2e-16	***
Personal Care	-1.8716218	0.0299893	-62.4097	< 2.2e-16	***
Waiting	-3.6080413	0.0674144	-53.5203	< 2.2e-16	***
Volunteering	-3.5948544	0.0669831	-53.6681	< 2.2e-16	***
Caregiving	-3.3621502	0.0598342	-56.1911	< 2.2e-16	***
Education	-5.0212337	0.1352830	-37.1165	< 2.2e-16	***
Work Production	-3.7153701	0.0710354	-52.3031	< 2.2e-16	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null deviance: 319859.4 on 61537 degrees of freedom

Residual deviance: 245959.8 on 61525 degrees of freedom

Chi-square: 73899.6 on 12 degrees of freedom, asymptotic p-value 0

AIC: 245983.8 AICC: 245983.9 BIC: 246092.2

Log posterior: -124191.7

Prior parameters: mu=0 sigma=100 nu=4

The Bayesian “ p values” here can be interpreted as two times the posterior probability that the true value of the associated parameter has a sign opposite that of the posterior mode (under an asymptotic z -approximation); this is equivalent to the result that would be obtained via a standard (frequentist) two-sided z -test of the hypothesis that the parameter value is zero, under the assumption that the posterior standard deviation is the standard error.

The results of this crude baseline model suggest that all endogenous events except Household Production and Leisure are significantly less likely to occur than Sleeping, and that Travel is not significantly more likely to occur than Sleeping. Note that, in the ordinal timing case of the egocentric relational event model, an intercept-only model such as this is equivalent to a series of Poisson models, and the sample means are approximately equal to the BPM estimates in this case:

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

	BPM pois.mle	
Caregiving	-3.3622	-3.3622
Communication	-0.7191	-0.7191
Eating	-0.0524	-0.0524
Education	-5.0212	-5.0212
Household Production	0.3306	0.3306
Leisure	0.5687	0.5687
Personal Care	-1.8716	-1.8716
Private personal care	-0.6843	-0.6843
Travel	0.0079	0.0079
Volunteering	-3.5949	-3.5949
Waiting	-3.6080	-3.6080
Work Production	-3.7154	-3.7154

3. Sequence statistics

While intercept-only models are interesting, the real strength of the **informR** tools comes from their ability to generate sufficient statistics to model complex event sequences using `rem()`. Here, we introduce the term *s-form* (or “sequence form”) to describe canonical sequences of events that we wish to model. Each s-form has a prefix and suffix. The prefix may consist of any combination of valid event types and represents the event sequence that precedes an event to be predicted. Correspondingly, the suffix should be a single event type representing the event to be predicted by the preceding events of the prefix. The suffix is *always* the final event type in the s-form framework.

For example, a simple s-form might consist of the event sequence $a \rightarrow b \rightarrow c$, which is read as “event *a* leads to event *b* which predicts event *c*” – or, less deterministically, “event *a* followed by event *b* predicts event *c*” Here, the prefix is “ $a \rightarrow b$ ” and the suffix is event “*c*.” As discussed below, it is possible to parameterize s-forms in several different ways using package **informR**.

The **informR** package uses simple regular expressions (regex) based on the alphabetic ids mapped to the event types in the `event.key` element in an `eventlist` object to represent s-forms. Two special regex operators are permitted in s-form expressions: the “|” character (alternation operator) and the “+” character (repetition operator) (Friedl 2006, pg. 18). The | operator is used to differentiate between two possible event paths and the + operator is used to indicate repetition, or persistence, of the preceding event. A single + operator may be nested within an | statement (see Section 3.1 on complex sequences, below, for example code) but nested | statements are not currently supported. The columns of Table 1 respectively report various useful canonical s-forms, their corresponding regex form, and their definitions.

Building sufficient statistics using the s-form framework is accomplished through the functions `gen.sformlist()` and `glb.sformlist()`. The first function, `gen.sformlist()`, can be used to construct statistics for specific sequences. For example, digram and trigram s-forms (e.g. $a \rightarrow b$ and $a \rightarrow b \rightarrow c$) are easily constructed using this function. The second function, `glb.sformlist()`, pools multiple s-form regular expressions into a single statistic. This is particularly handy for modeling, say, a class of behaviors that is characterized by multiple s-forms sharing a common feature. Both functions are wrappers for an internal function, `gen.sform()`, which has limited functionality to the user. The mandatory arguments for



s-form	regex	Definition
$a \rightarrow a$	aa	Inertial term: s-form of the type “event a predicts event a ”.
$a \rightarrow b$	ab	Basic digram transition term: s-form of the type “event a predicts event b ”.
	$a + b$	Transition term with persistence: s-form of the type “some series of events a predicts event b ”.
$a \rightarrow b \rightarrow c$	abc	Basic trigram transition term: s-form of the type “event a followed by event b predicts event c ”.
	$(a b)c$	Transition term with disjunction: s-form of the type “event a OR event b predicts event c ”.

Table 1: Examples of canonical s-forms, s-form regular expression representations, and s-form definitions for modeling relational events using **informR** and **rem()**.

`gen.sformlist` are: `evl`, an eventlist object, and `sforms`, a character vector of regex s-forms. The mandatory arguments for `glb.sformlist` are: `evl`, an eventlist object, `sforms`, a list of character vectors of regex s-forms, and `new.names`, which is a character vector of variable names that has the same length as `sforms`. Additional optional arguments for both functions include: `cond = FALSE`, `interval = FALSE`, and `warn = TRUE`, which I will discuss below. Using the examples provided in Table 1, generating sufficient statistics for modeling inertial (or persistence) effects of the endogenous events is done by:

```
R> a1 <- paste(evls$event.key[-9, 1], evls$event.key[-9, 1], sep = "")
R> beta.sforms <- gen.sformlist(evls, a1)
```

The `beta.sforms` object is a list of three dimensional arrays, the length of which is equal to the number of valid event histories in the dataset (i.e., the length of `evls$eventlist`). Each element of `beta.sforms` consists of an i, j, k array where i indexes the event, j indexes the event type, and k indexes the sufficient statistic to a corresponding element passed to `gen.sformlist` by `sforms`. Examining the first event of the first element of `beta.sforms` reveals the structure of the output:

```
R> beta.sforms[[1]][1, , ]
```

```
   aa bb cc dd ee ff gg hh jj kk ll mm nn
a  0  0  0  0  0  0  0  0  0  0  0  0  0
b  0  0  0  0  0  0  0  0  0  0  0  0  0
c  0  0  0  0  0  0  0  0  0  0  0  0  0
d  0  0  0  0  0  0  0  0  0  0  0  0  0
e  0  0  0  0  0  0  0  0  0  0  0  0  0
f  0  0  0  0  0  0  0  0  0  0  0  0  0
g  0  0  0  0  0  0  0  0  0  0  0  0  0
h  0  0  0  0  0  0  0  0  0  0  0  0  0
```

```

j 0 0 0 0 0 0 0 0 0 0 0 0 0
k 0 0 0 0 0 0 0 0 0 0 0 0 0
l 0 0 0 0 0 0 0 0 0 0 0 0 0
m 0 0 0 0 0 0 0 0 0 0 0 0 0
n 0 0 0 0 0 0 0 0 0 0 0 0 0

```

which shows that the row and column labels of each event matrix correspond to the alphabetic ids of the event types in `event.key` and their inertial s-forms, respectively. For completeness, it is useful to walk through the relationship between the inertial sufficient statistics and the event history. Let us take the first three events from the event history associated with `atus80ord` respondent “20030101033341” as our example.

```
R> evls$eventlist$"20030101033341"
```

```

[1] 1 2 2 7 1
attr("char")
[1] "a" "b" "b" "g" "a"

```

```
R> sapply(attr(evls$eventlist$"20030101033341", "char")[1:3],
  sf2nms, event.key = evls$event.key)
```

```

      a      b      b
"Sleeping" "Eating" "Eating"

```

This subsequence consists of the actor sleeping followed by two instances of eating. The `sf2nms()` is a convenience function that substitutes the character representation of the event types with their respective names. The corresponding inertial s-forms for these two event types are $a \rightarrow a$, $b \rightarrow b$. These are located in the `beta.sforms$"20030101033341"` `sformlist`:

```
R> beta.sforms$"20030101033341"[1:4, , c("aa", "bb")]
```

```
, , aa
```

```

  a b c d e f g h j k l m n
1 0 0 0 0 0 0 0 0 0 0 0 0 0
2 1 0 0 0 0 0 0 0 0 0 0 0 0
3 0 0 0 0 0 0 0 0 0 0 0 0 0
4 0 0 0 0 0 0 0 0 0 0 0 0 0

```

```
, , bb
```

```

  a b c d e f g h j k l m n
1 0 0 0 0 0 0 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0 0 0 0 0 0 0
3 0 1 0 0 0 0 0 0 0 0 0 0 0
4 0 1 0 0 0 0 0 0 0 0 0 0 0

```

Thus, the inertial statistics relate back to the eventlist in the following way: the first event has alphabetic id “a”, so the “a” column in the next event row (row 2) is equal to 1 in the “aa” element of the array. Likewise, the second and third events have alphabetic id “b,” so the “b” column in the third and fourth rows are set equal to 1 in the “bb” element of the array. This is the general framework for the underlying statistic construction methods of the **informR** package.

To use the newly created sufficient statistics in a relational event model, the `beta.sforms` `sformlist` object needs to be appended to a `statslist`. This is easily done using the `slbind()` function, which takes an `sformlist` and a `statslist` in arguments `sformstats` and `statslist`, respectively. Optionally, `slbind()` can “translate” the alphabetic ids back into event type names (i.e., from “bb” to “EatingEating”) if optional arguments `new.names = TRUE` and `event.key = evls$event.key` are passed to it. For convenience, the function `sfl2statslist()` can be used to directly convert an `sformlist` object to a `statslist` (i.e., without appending it to an already existing `statslist` as above), though name translation is not currently supported.

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

The new `statslist`, `beta.ints`, can now be directly passed to `rem()`:

```
R> beta.fit <- rem(evls$eventlist, beta.ints, estimator = "BPM",
+   prior.param = list(mu = 0, sigma = 100, nu = 4))$coef
R> round(cbind(beta.fit[13:25]), 4)
```

	[,1]
SleepingSleeping	-1.0863
EatingEating	-3.8336
Private personal carePrivate personal care	-2.5093
Household ProductionHousehold Production	-0.0121
TravelTravel	-0.3470
CommunicationCommunication	0.4067
LeisureLeisure	-0.1748
Personal CarePersonal Care	0.3683
WaitingWaiting	1.6504
VolunteeringVolunteering	3.2567
CaregivingCaregiving	3.2422
EducationEducation	-8.3913
Work ProductionWork Production	2.9646

The inertial statistics suggest that, for the older US population, most spells of activity are not likely to be immediately repeated, net of the overall tendency for the event to occur at all. The exceptions, obviously, include communication, personal care, waiting, volunteering, caregiving, and work production. The conditional probabilities of each activity spell being followed by exactly the same type of activity spell can be derived in the usual way from Equation 4. Consider, for example, the conditional probability that a spell of communication is immediately followed by another spell of communication. Given that the most recent event

was a communication event, only the `CommunicationCommunication` sequence term has a satisfied prefix and is hence active (adding 0.4067 to the log hazard of a `Communication` event). The conditional probability of the next event being a communication spell is then equal to the total hazard of a communication event (the baseline times the digram effect), divided by the total hazard of all events (the communication hazard plus the baseline hazards for all other events). This can be computed for the above model as follows:

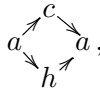
```
R> comComHaz <- exp(sum(beta.fit[c(5, 18)]))
R> all0thHaz <- exp(beta.fit[(1:12)[-5]])
R> comComHaz / sum(comComHaz + all0thHaz)
```

```
[1] 0.05074562
```

Hence, we predict an approximately 5% chance of the next event being a communication spell given a `Communication` event (small, but about 1.5 times the base rate for this spell type).

3.1. Constructing complex sequences

The inertial s-forms provide a relatively simple example of how to construct sequence statistics for `rem()`. Let us now consider a couple more complex cases using some gerontological theories about the older US population to motivate our statistics. For example, the older population is believed to have their sleep frequently interrupted by the need to use the bathroom or perform some other personal care activity, or interrupted in order to help another older person (usually a spouse) with similar needs (Barker and Mitteness 1988). These types of activities are captured in the data by “Private personal care”, “Personal Care”, and “Caregiving” spell types. Without loss of generality, we can use the `|` operator to combine information from the two slightly different personal care spell types in our s-form construction. Thus, we can model the likelihood of sleep being interrupted by personal care by using the s-form:



and the likelihood of sleep being interrupted by providing care with the s-form: $a \rightarrow l \rightarrow a$. We use the persistence `+` operator after the “`l`” in the latter s-form because the inertial effect of caregiving was very large and significant in the previously fit model (`beta.fit`, above), which gives us reason to suspect that if caregiving is going to interrupt sleep, it will do so as a sequence of caregiving spells rather than a single instance. Specifying the special operators causes `gen.sformlist()` to use a slower algorithm for identifying valid subsequences and a note is issued:

```
R> a2 <- c("a(c|h)a", "al+a")
R> gamma.sforms <- gen.sformlist(evls, a2)
```

Note:

```
a(c|h)a S-form(s) contains special regex syntax.
Using slow search methods.
```

Note:

```
al+a S-form(s) contains special regex syntax.
Using slow search methods.
```

```
R> gamma.ints <- slbind(gamma.sforms, beta.ints, new.names = TRUE,
+   event.key = evls$event.key)
R> gamma.fit <- rem(evls$eventlist, gamma.ints, estimator = "BPM",
+   prior.param = list(mu = 0, sigma = 100, nu = 4))
R> summary(gamma.fit)
```

Egocentric Relational Event Model (Ordinal Likelihood)

	Post.Mode	Post.SD	Z value	Pr(> z)	
Eating	0.100907	0.016147	6.2495	4.118e-10	***
Private personal care	-0.787081	0.019822	-39.7081	< 2.2e-16	***
Household Production	0.339104	0.015493	21.8873	< 2.2e-16	***
Travel	0.052881	0.016301	3.2440	0.001179	**
Communication	-0.746241	0.020027	-37.2617	< 2.2e-16	***
Leisure	0.616306	0.014958	41.2014	< 2.2e-16	***
Personal Care	-2.036929	0.030837	-66.0543	< 2.2e-16	***
Waiting	-3.621061	0.068077	-53.1903	< 2.2e-16	***
Volunteering	-3.672055	0.069781	-52.6227	< 2.2e-16	***
Caregiving	-3.432682	0.064196	-53.4723	< 2.2e-16	***
Education	-5.019067	0.135321	-37.0901	< 2.2e-16	***
Work Production	-3.766753	0.073064	-51.5542	< 2.2e-16	***
SleepingSleeping	-0.848855	0.066511	-12.7626	< 2.2e-16	***
EatingEating	-3.858526	0.189726	-20.3373	< 2.2e-16	***
Private personal carePriva...	-2.27346	0.201444	-11.2858	< 2.2e-16	***
Household ProductionHouseh...	-0.040021	0.027019	-1.4812	0.138548	
TravelTravel	-0.371463	0.038617	-9.6192	< 2.2e-16	***
CommunicationCommunication	0.386483	0.058181	6.6428	3.078e-11	***
LeisureLeisure	-0.208772	0.023220	-8.9909	< 2.2e-16	***
Personal CarePersonal Care	0.556270	0.171652	3.2407	0.001192	**
WaitingWaiting	1.632468	0.508972	3.2074	0.001339	**
VolunteeringVolunteering	3.238798	0.255101	12.6961	< 2.2e-16	***
CaregivingCaregiving	3.214235	0.211677	15.1846	< 2.2e-16	***
EducationEducation	-4.440779	38.358737	-0.1158	0.907835	
Work ProductionWork Production	2.946673	0.318480	9.2523	< 2.2e-16	***
Sleeping(Priv... Pers...)Sleeping	1.215852	0.031546	38.5421	< 2.2e-16	***
SleepingCaregiving+Sleeping	-0.370198	0.267511	-1.3839	0.166401	

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Null deviance: 315678.6 on 61537 degrees of freedom
Residual deviance: 241397.7 on 61510 degrees of freedom
Chi-square: 74280.92 on 27 degrees of freedom, asymptotic p-value 0
AIC: 241451.7 AICC: 241451.7 BIC: 241695.4
Log posterior: -123425.3
Prior parameters: mu=0 sigma=100 nu=4
```

The results suggest that, ceteris paribus, spells of sleep tend to be interrupted by personal care

activities ($\beta = 1.216^{***}$) but not by caregiving ($\beta = -0.370^{n.s.}$). However, overall tendencies for sleep interruption to occur from any activity are unaccounted for in this model. To control for this, we could construct individual sufficient statistics for all possible trigrams (or, if theoretically meaningful, higher order sequences) and estimate their individual effects. Alternatively, we could pool those trigrams into a single sufficient statistic and save degrees of freedom. To accomplish this, use the `glb.sformlist()` function:

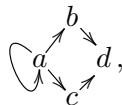
```
R> sleep.sfs <- paste("a", letters[2:14], "a", sep = "")
R> sleep.sforms <- glb.sformlist(evls, sforms = list(sleep.sfs),
+   new.names = "InterSleep")
R> sleep.ints <- slbind(sleep.sforms, gamma.ints)
R> sleep.fit <- rem(evls$eventlist, sleep.ints, estimator = "BPM",
+   prior.param = list(mu = 0, sigma = 100, nu = 4))
R> round(cbind(BPM = sleep.fit$coef, Z = sleep.fit$coef /
+   sleep.fit$sd)[26:28, ], 4)
```

	BPM	Z
Sleeping(Private personal care Personal Care)Sleeping	1.5334	44.7723
SleepingCaregiving+Sleeping	-0.2575	-0.9647
InterSleep	-1.2908	-22.2002

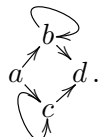
Interestingly, the negative coefficient for the overall tendency for sleep interruption trigrams is negative (-1.291^{***}), strongly suggesting that interruptions to sleep are not likely. That is, whatever happens following a spell of sleep is not likely to result in more sleeping. When sleep interruptions do occur, however, they are most likely to be the result of some personal care activity.

4. Additional s-form parametrizations

The preceding exercise demonstrates how the **informR** package can handle `statslist` construction of most combinations of canonical s-forms. For example, consider a relatively complex s-form:



which contains both a persistence term and a divergence term preceding the suffix and which would be passed as “ $a + (b|c)d$.” One exception to this is when an s-form contains more than one persistence term inside a divergence statement, such as:



To obtain a properly formed sufficient statistic for such cases, the best strategy is to write down each possible valid outcome of the s-form regex and pass them to `glb.sformlist` with the `sform` parameter. Using this s-form as an example, there are two possible paths from “ a ” to “ d ” that the internal `gen.ev1` function can interpret. These are expressed by the following s-form regexes: “ $a(b + |c)d$ ” and “ $a(b|c+)d$.”


```
R> tmp.sforms <- glb.sformlist(evls, list(c("a(b+|c)d", "a(b|c+)d")),
+   new.names = "a(b+|c+)d")
```

Note:

a(b+|c)d S-form(s) contains special regex syntax.
Using slow search methods.

Note:

a(b|c+)d S-form(s) contains special regex syntax.
Using slow search methods.

```
R> evls$eventlist[[1]]
```

```
[1] 1 2 3 4 2 4 1
attr(,"char")
[1] "a" "b" "c" "d" "b" "d" "a"
```

Examine the sformlist output that corresponds with the “ $a(b + |c+)d$ ” s-form:

```
R> tmp.sforms[[1]]
```

```
, , a(b+|c+)d

  a b c d e f g h j k l m n
1 0 0 0 0 0 0 0 0 0 0 0 0 0
2 0 1 1 0 0 0 0 0 0 0 0 0 0
3 0 1 0 1 0 0 0 0 0 0 0 0 0
4 0 0 0 0 0 0 0 0 0 0 0 0 0
5 0 0 0 0 0 0 0 0 0 0 0 0 0
6 0 0 0 0 0 0 0 0 0 0 0 0 0
7 0 0 0 0 0 0 0 0 0 0 0 0 0
```

From the example output, we see that the first three events in the event history are “a”, “b”, “c” (corresponding to spells of sleeping, eating, then personal care activities). This selected sub-sequence of the event history, not coincidentally, fulfills one of the many possible outcomes from the complex s-form that we specified. Following our prior discussion on the relationship between the s-form, the event history, and the statslist, we can see that since “a” occurs the sufficient statistic for the s-form predicts that “b” or “c” will happen next and both respective columns are incremented in the second event row of the statslist. The second event is “b”, and since we predict *at least one* “b” with the “+” character in the s-form, both “b” and the suffix event “d” columns are incremented in the third event row. Event “c” is no longer a predicted outcome because “c” did not occur after “a.” Recall that there may be more than one way to specify s-form regexes. Here, we could have achieved the same result with `c("ab+d", "ac+d")` in the `sform` parameter.

Creative use of the above strategy may provide a work-around for many cases that are not natively supported by package **informR**, such as s-forms with nested disjunctions.

4.1. Conditioning out the prefix

The default settings of `gen.sformlist()` and `glb.sformlist()` produce sufficient statistics that allow the model to influence the hazards of each non-initial element in the s-form, given what has happened up to that point in the event history. For example, an s-form term $a \rightarrow b \rightarrow c$ will add its corresponding parameter value to the hazard of b given that a has just transpired *and* to the hazard of c given that $a \rightarrow b$ has just transpired. Thus, this term's contribution to the likelihood of the event history grows larger as events of a occur and the subsequence of events $a \rightarrow b$ occur. For certain applications, however, it is useful to estimate only the hazard of the suffix event given the preceding events (i.e., without realizing contributions to the likelihood from any prefix elements). To accomplish this, set `cond = TRUE` in `gen.sformlist()` and `glb.sformlist()`.

The names of the s-forms in the output will be slightly different when using `gen.sformlist` with `cond = TRUE` as the prefix is enveloped between parentheses to indicate a conditional s-form. Here is an example comparing the $a \rightarrow b \rightarrow c$ s-form with and without the prefix contributions to the likelihood:

```
R> abc.sform <- gen.sformlist(evls, c("abc"), cond = FALSE)
R> abc.cond.sform <- gen.sformlist(evls, c("abc"), cond = TRUE)
R> abc.sform[[1]]
```

```
, , abc
```

	a	b	c	d	e	f	g	h	j	k	l	m	n
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0

```
R> abc.cond.sform[[1]]
```

```
, , (ab)c
```

	a	b	c	d	e	f	g	h	j	k	l	m	n
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0

It should be clear that, in the case of simple digrams, there is no distinction between $a \rightarrow b$ and $(a) \rightarrow b$ because only the hazard of the suffix event is affected in both cases. By contrast,

$a \rightarrow b \rightarrow c$ and $(a \rightarrow b) \rightarrow c$, are different: specifically, the former term is equivalent to the combination of terms $(a) \rightarrow b$ and $(a \rightarrow b) \rightarrow c$ with the parameters for the latter constrained to be equal. Unconditional s-forms are thus a parsimonious way to capture the tendency for a given chain of events to take place, while conditional s-forms allow for the specification of terms that affect only the end of a complex event sequence.

4.2. Modifying statslists

The above examples illustrate how easy it is to build complex `statslists` using the `informR` package, including how multiple statslist models may be combined using the `slbind()` function. This section demonstrates how to modify statslists to create new models by dropping and adding elements. Dropping elements of statslists is easy using the `sldrop()` function. This function takes three arguments, `statslist`, `varname`, and `type` and returns a new `statslist` that omits any sufficient statistic column matching any name in `varname`. The `type` argument is used to indicate whether the new `statslist` should be “global” or “local” – as in previously discussed methods.

Our example will create s-forms to test for communication-terminated sequences of events (i.e., sequences of events that ultimately lead to a `Communication` spell):

```
R> sforms <- c("bf", "ef", "gf", "cf", "egf", "cef", "af")
R> delta.sforms <- gen.sformlist(evls, sforms)
R> delta.ints <- slbind(delta.sforms, alpha.ints)
R> delta.fit <- rem(evls$eventlist, delta.ints)
R> summary(delta.fit)
```

Egocentric Relational Event Model (Ordinal Likelihood)

	Post.Mode	Post.SD	Z value	Pr(> z)	
Eating	-0.0524428	0.0156947	-3.3414	0.0008335	***
Private personal care	-0.6843113	0.0189127	-36.1827	< 2.2e-16	***
Household Production	0.3306127	0.0143563	23.0291	< 2.2e-16	***
Travel	-0.0029505	0.0157808	-0.1870	0.8516886	
Communication	-0.9306083	0.0315665	-29.4809	< 2.2e-16	***
Leisure	0.5750562	0.0141408	40.6664	< 2.2e-16	***
Personal Care	-1.8716223	0.0299893	-62.4097	< 2.2e-16	***
Waiting	-3.6080436	0.0674145	-53.5203	< 2.2e-16	***
Volunteering	-3.5948566	0.0669832	-53.6681	< 2.2e-16	***
Caregiving	-3.3621520	0.0598343	-56.1911	< 2.2e-16	***
Education	-5.0212453	0.1352840	-37.1163	< 2.2e-16	***
Work Production	-3.7153727	0.0710355	-52.3030	< 2.2e-16	***
bf	0.2833377	0.0533442	5.3115	1.087e-07	***
ef	0.7859901	0.0463006	16.9758	< 2.2e-16	***
gf	0.3104193	0.0440410	7.0484	1.810e-12	***
cf	-0.3320642	0.0864922	-3.8392	0.0001234	***
egf	-0.0497697	0.0276494	-1.8000	0.0718562	.
cef	0.1498079	0.0418826	3.5769	0.0003478	***
af	-0.4283528	0.0821778	-5.2125	1.863e-07	***

```

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Null deviance: 315678.6 on 61537 degrees of freedom
Residual deviance: 245476.3 on 61518 degrees of freedom
    Chi-square: 70202.27 on 19 degrees of freedom, asymptotic p-value 0
AIC: 245514.3 AICC: 245514.3 BIC: 245685.8
Log posterior: -141756.8
Prior parameters: mu=0 sigma=1000 nu=4

```

The coefficients for communication preceded by travel leads to leisure ("egf") and "Travel" are not significant. We can drop the sufficient statistics for these terms from the model using the `sldrop()` function and then compare the models by BIC:

```

R> delta.ints2 <- sldrop(delta.ints, c("egf", "Travel"))
R> delta.fit2 <- rem(evls$eventlist, delta.ints2)
R> names(delta.fit2$coef)

```

```

 [1] "Eating"           "Private personal care" "Household Production"
 [4] "Communication"   "Leisure"              "Personal Care"
 [7] "Waiting"         "Volunteering"         "Caregiving"
[10] "Education"       "Work Production"      "bf"
[13] "ef"              "gf"                   "cf"
[16] "cef"             "af"

```

```

R> c(delta.fit$BIC, delta.fit2$BIC)

```

```

[1] 245685.8 245667.1

```

The more parsimonious model is preferred by BIC.

The function `slbind.cond()` allows to add interaction variables to a `statslist` object. The `slbind.cond()` function adds actor-by-event conditional or interaction variables to a `statslist` object. This function is useful when an event history, or part thereof, depends upon some variable (actor characteristics, temporality, etc). The function takes seven arguments: `intvar`, a numeric variable; `statslist`, the `statslist` object; `var.suffix`, a character string to append to the name of the new variable; `sl.ind`, the column indexes of the events to be interacted with the `intvar` in each matrix element of the `statslist`; `who.evs`, for local statistic construction, the indexes of the actors to receive the new variable; and, `type`, an indicator for local or global statistics. Additional arguments can be passed to `abind()` using R's ellipsis arguments (`...`).

In the following example, a model that interacts an indicator for whether or not the respondent is female with the terms in the sleep interruption s-forms modeled above is fit to the data. This is motivated by the hypothesis that older males, due to declining bladder and prostate function, are more likely to have their sleep interrupted to urinate (Tikkinen, Tammela, Huhtalal, and Auvinen 2006) and that older females are more likely to be caretakers of their afflicted spouses (Willette-Murphy, Todera, and Yeaworth 2006).

```

R> sl.ind <- 26:27
R> fem.evs <- unlist(glapply(atus80ord$SEX, atus80ord$TUCASEID, unique,
+   regroup = FALSE))
R> fem.evs <- ifelse(fem.evs == 1, 1, 0)
R> gamma.ints2 <- slbind.cond(fem.evs, gamma.ints, sl.ind = sl.ind,
+   var.suffix = "FEM")
R> sl.ind <- 26:29
R> gamma.fit2 <- rem(evls$eventlist, gamma.ints2, estimator = "BPM",
+   prior.param = list(mu = 0, sigma = 100, nu = 4))
R> round(cbind(BPM = gamma.fit2$coef[sl.ind], Z = gamma.fit2$coef[sl.ind] /
+   gamma.fit2$sd[sl.ind]), 4)

```

	BPM	Z
Sleeping(Private personal care Personal Care)Sleeping	1.2195	32.9940
SleepingCaregiving+Sleeping	-0.5895	-1.6327
Sleeping(Private personal care Personal Care)Sleeping.FEM	-0.0120	-0.1929
SleepingCaregiving+Sleeping.FEM	0.5522	1.0512

Despite the gender interaction coefficients being in the hypothesized direction, the effects are not statistically significant in this model. Older men are not more likely to have their sleep interrupted for personal care activities and older women are not more likely to have their sleep interrupted to perform caregiving activities in this population.

5. Using interval time data

The `rem()` function can fit models for sequences of events when the exact (or nearly exact) timing of the events is known by providing the appropriate data in `eventlist` and setting the `timing` argument to “interval.” In general, generating s-form sufficient statistics for interval time models is the same as the ordinal time case, though there are subtle differences. We will review how to construct s-forms for the interval time likelihood relational event model in this section.

First, recall that the interval time data must contain three columns where the first column indexes the events, the second column indexes the temporal information of when the event transpired, and the third column indexes an event history grouping factor. In the example data, `atus80int`, an “event” is recorded when an actor either starts or stops a particular activity spell. Eventlist objects and statslist objects for intercept models are created for this data using `gen.evl()` and `gen.intercepts()`, respectively, but with the interval time data arguments needing to be specified. As the interval timing models take slightly longer to fit, the example uses only a random subset ($N = 500$) of the data:

```

R> evlsint <- gen.evl(atus80int[, 1:3])
R> set.seed(570819)
R> evlsint$eventlist <- evlsint$eventlist[sample(1:length(evlsint$eventlist),
+   500)]
R> evlsint$event.key

```

```

      id event.type
[1,] "a" "Sleeping|START"
[2,] "b" "Sleeping|STOP"
[3,] "c" "Eating|START"
[4,] "d" "Eating|STOP"
[5,] "e" "Private personal care|START"
[6,] "f" "Private personal care|STOP"
[7,] "g" "Household Production|START"
[8,] "h" "Household Production|STOP"
[9,] "i" "Travel|START"
[10,] "j" "Travel|STOP"
[11,] "k" "Communication|START"
[12,] "l" "Communication|STOP"
[13,] "m" "Leisure|START"
[14,] "n" "Leisure|STOP"
[15,] "o" "Personal Care|START"
[16,] "p" "Personal Care|STOP"
[17,] "q" "NA|START"
[18,] "r" "NA|STOP"
[19,] "s" "Waiting|START"
[20,] "t" "Waiting|STOP"
[21,] "u" "Volunteering|START"
[22,] "v" "Volunteering|STOP"
[23,] "w" "Caregiving|START"
[24,] "x" "Caregiving|STOP"
[25,] "y" "Education|START"
[26,] "z" "Education|STOP"
[27,] "A" "Work Production|START"
[28,] "B" "Work Production|STOP"

```

```
R> evlsint$eventlist$"20040706041558"
```

```

      [,1]      [,2]
[1,]    1    0.000
[2,]    2  240.000
[3,]    3  240.001
[4,]    4  300.000
[5,]    7  300.001
[6,]    8  310.000
[7,]   13  310.001
[8,]   14  340.000
[9,]   13  340.001
[10,]  14  450.000
[11,]    3  450.001
[12,]    4  510.000
[13,]    1  510.001
[14,]    2  570.000

```

```

[15,] 13 570.001
[16,] 14 870.000
[17,] 13 870.001
[18,] 14 1020.000
[19,] 1 1020.001
[20,] 2 1650.001
attr(,"char")
 [1] "a" "b" "c" "d" "g" "h" "m" "n" "m" "n" "c" "d" "a" "b" "m" "n" "m" "n"
[19] "a" "b"

```

```
R> int.base <- gen.intercepts(evlsint, type = 1, contr = FALSE)
```

Note that we now have twice as many event types as we did during the ordinal data example. Each event list in `evlsint$eventlist` is now a two-column matrix where the first column indexes the numeric code of the event type and the second column indexes the timing information. Failing to set `contr = FALSE` in `gen.intercepts()` in the interval case may result in an improperly identified model.

Before we can proceed with fitting the intercepts-only model, we need to consider that not all sequences of events are possible – the event support, $\mathbb{A}(A_t)$, is not in this case fixed. Since events in the present formulation indicate onsets and termini of activity spells, the most trivial constraint we must enforce is that a **START** event for a given activity cannot occur if there has been a previous **START** event for that activity without the most recent event of that activity being a **STOP** spell (i.e., we cannot initiate an activity in which we are already engaged). Likewise, we cannot terminate an activity without starting it: a **STOP** event for a given activity cannot occur unless the most recent event for that activity is a **START** event. Likewise, there are obvious physical constraints that should induce additional restrictions on event ordering, such as the fact that one cannot, e.g., initiate eating without first ceasing to sleep. In the particular case of the ATUS, many of these issues are simplified by the nature of the study, which defines activities such that subjects can be engaged in at most one activity at any given time, and that activity begins with the onset of the first spell of the day. This amounts to a blanket constraint that (1) no **START** event can occur unless the previous event was a **STOP** event and (2) the only event that can immediately follow a **START** event is a **STOP** event of the same activity type. `rem()` allows for the specification of arbitrary constraints of this type, using the `supplist` argument (which allows users to specify at each point in the event history those events that were at risk for occurring). As documented in `help("rem")`, `supplist` is defined as follows:

If desired, support constraints for the event histories can be specified using `supplist`. `supplist` should be a list with one element per history, each of which should be an event order by event type logical matrix. The ij th cell of this matrix should be **TRUE** if an event of type j was a possible next event given the preceding code $i - 1$ events, and **FALSE** otherwise. (By default, all events are assumed to be possible at all times.) As with the model statistics, the elements of the support list must be user supplied, and will often be history-dependent. (E.g., in a model for spell-based data, event types will come in onset/termination pairs, with terminal events necessarily being preceded by corresponding onset events.) (Butts 2015)

Failure to specify support constraints where applicable can badly bias coefficient estimates, and it is hence important that they be considered when fitting complex relational event models.

Turning back to our example data, we can create a support list satisfying the constraints of the ATUS data with the following R code:

```
R> eT <- evlsint$event.key[, 2]
R> supplist <- list()
R> for (i in 1:length(evlsint$eventlist)) {
+   supplist[[i]] <- matrix(0, nrow = NROW(evlsint$eventlist[[i]]),
+     ncol = length(eT))
+   colnames(supplist[[i]]) <- eT
+ }
R> for (i in 1:length(evlsint$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
+     }
+   }
+ }
+ }
```

Now we can fit a new baseline model by passing `supplist = supplist` to `rem()`:

```
R> intfit.base <- rem(evlsint$eventlist, int.base, supplist = supplist,
+   timing = "interval", estimator = "BPM")
R> summary(intfit.base)
```

Egocentric Relational Event Model (Interval Likelihood)

	Post.Mode	Post.SD	Z value	Pr(> z)
Sleeping START	4.967864	0.028421	174.7954	< 2.2e-16 ***
Sleeping STOP	-5.690678	0.028421	-200.2278	< 2.2e-16 ***
Eating START	4.900197	0.029399	166.6787	< 2.2e-16 ***
Eating STOP	-3.589547	0.029399	-122.0974	< 2.2e-16 ***
Private personal care START	4.282762	0.040032	106.9833	< 2.2e-16 ***
Private personal care STOP	-3.497837	0.040032	-87.3759	< 2.2e-16 ***
Household Production START	5.272565	0.024405	216.0465	< 2.2e-16 ***
Household Production STOP	-3.760817	0.024405	-154.1017	< 2.2e-16 ***
Travel START	4.934185	0.028904	170.7114	< 2.2e-16 ***
Travel STOP	-2.790073	0.028904	-96.5301	< 2.2e-16 ***

Communication START	4.256786	0.040555	104.9624	< 2.2e-16	***
Communication STOP	-3.899224	0.040555	-96.1457	< 2.2e-16	***
Leisure START	5.509079	0.021683	254.0755	< 2.2e-16	***
Leisure STOP	-4.653257	0.021683	-214.6055	< 2.2e-16	***
Personal Care START	3.217249	0.068199	47.1741	< 2.2e-16	***
Personal Care STOP	-3.998140	0.068199	-58.6242	< 2.2e-16	***
NA START	2.698641	0.088388	30.5316	< 2.2e-16	***
NA STOP	-4.242211	0.088388	-47.9951	< 2.2e-16	***
Waiting START	1.343119	0.174078	7.7156	1.199e-14	***
Waiting STOP	-3.856553	0.174078	-22.1542	< 2.2e-16	***
Volunteering START	1.213907	0.185695	6.5371	6.273e-11	***
Volunteering STOP	-4.582844	0.185695	-24.6794	< 2.2e-16	***
Caregiving START	1.430130	0.166667	8.5808	< 2.2e-16	***
Caregiving STOP	-4.013357	0.166667	-24.0801	< 2.2e-16	***
Education START	-0.207479	0.377964	-0.5489	0.583	
Education STOP	-4.302118	0.377964	-11.3823	< 2.2e-16	***
Work Production START	1.430130	0.166667	8.5808	< 2.2e-16	***
Work Production STOP	-4.863244	0.166667	-29.1795	< 2.2e-16	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null deviance: 174603.4 on 18228 degrees of freedom

Residual deviance: 21536.87 on 18200 degrees of freedom

Chi-square: 153066.6 on 28 degrees of freedom, asymptotic p-value 0

AIC: 21592.87 AICC: 21592.96 BIC: 21811.57

Log posterior: -38795.9

Prior parameters: mu=0 sigma=1000 nu=4

Per Equation 1, the coefficients for the intercept model are estimates of the log hazards ($\log \lambda$) for each event type, at each point in the event history for which the corresponding event type is possible (i.e., in $A(A_t)$). Under this model (and due to the exclusivity constraint of the ATUS), the starting event coefficient for a given activity is proportional to the log probability that any given spell in the event history will be of the corresponding activity type. This can be verified by simple tabulation:

```
R> bpm.start <- (exp(intfit.base$coef[grep("START",
+   names(intfit.base$coef))]) /
+   sum(exp(intfit.base$coef[grep("START", names(intfit.base$coef))])))
R> in.ids <- which(atus80int$TUCASEID %in% names(evlsint$eventlist) &
+   grepl("START", atus80int$Events))
R> mle.start <- prop.table(table(atus80int[in.ids, "Events"]))
R> round(cbind(BPM = sort(bpm.start), MLE = sort(mle.start)), 4)
```

	BPM	MLE
Education START	0.0008	0.0008
Volunteering START	0.0032	0.0032
Waiting START	0.0036	0.0036
Caregiving START	0.0039	0.0039

Work Production START	0.0039	0.0039
NA START	0.0140	0.0140
Personal Care START	0.0236	0.0236
Communication START	0.0667	0.0667
Private personal care START	0.0685	0.0685
Eating START	0.1269	0.1269
Travel START	0.1313	0.1313
Sleeping START	0.1358	0.1358
Household Production START	0.1842	0.1842
Leisure START	0.2334	0.2334

By turns, the inverse exponents of the stopping event coefficients can here be interpreted as the average duration of a particular spell of activity. For example, the average length of a spell of sleeping in this population is $\frac{1}{e^{-5.690678}} = 296.0943$ minutes, or roughly 5 hours. This follows from the piecewise constant hazard assumption, which (together with the support constraints) results in an exponential waiting time distribution for STOP events following a corresponding START event.

Sequence statistics for interval time models are created and manipulated in the usual way (see above for details). Caution should be taken to ensure that sequences of events are properly specified in the s-form to include the onset and termination of each spell. Here, we illustrate the power of `rem()` to estimate highly complex conditional sequence statistics with the help of package **informR** via our sleep interruption example. Specifically, we will construct a model with terms for sleep interrupted by any other spell of activity and by personal care spells. We are defining the sleep interruption effects to be conditional on (1) sleeping, then (2) waking, then (3) doing something else, and finally, (4) going back to sleep. The sufficient statistics will be parametrized to model both the onset of sleeping and the duration of the spell, given that sleep followed by the other activities had previously occurred. This may be represented by a pair of s-forms: (*Sleeping|Start* → *Sleeping|Stop* → *Something|Start* → *Something|Stop*) → *Sleeping|Start*, for the onset of a conditional sleeping spell, and (*Sleeping|Start* → *Sleeping|Stop* → *Something|Start* → *Something|Stop* → *Sleeping|Start*) → *Sleeping|Stop* for the duration of the conditional spell. To construct the model, we need to use `glb.sformlist()` as both s-forms pool information from multiple spells into single elements of the statistic:

```
R> sleepA <- paste("ab", c(letters[seq(1, 26, 2)], "A"),
+   c(letters[seq(2, 26, 2)], "B"), "a", sep = "")
R> sleepB <- paste("ab", c(letters[seq(1, 26, 2)], "A"),
+   c(letters[seq(2, 26, 2)], "B"), "ab", sep = "")
R> 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"))
R> sleep.int <- slbind(sleep.glbs, int.base)
R> intfit.sleep <- rem(evlsint$eventlist, sleep.int, supplist = supplist,
+   timing = "interval", estimator = "BPM")
R> summary(intfit.sleep)
```

Egocentric Relational Event Model (Interval Likelihood)

	Post.Mode	Post.SD	Z value	Pr(> z)	
Sleeping START	4.989059	0.029361	169.9212	< 2.2e-16	***
Sleeping STOP	-5.707896	0.029248	-195.1566	< 2.2e-16	***
Eating START	4.900197	0.029399	166.6787	< 2.2e-16	***
Eating STOP	-3.589547	0.029399	-122.0974	< 2.2e-16	***
Private personal care START	4.282762	0.040032	106.9833	< 2.2e-16	***
Private personal care STOP	-3.497837	0.040032	-87.3759	< 2.2e-16	***
Household Production START	5.272565	0.024405	216.0465	< 2.2e-16	***
Household Production STOP	-3.760817	0.024405	-154.1017	< 2.2e-16	***
Travel START	4.934185	0.028904	170.7114	< 2.2e-16	***
Travel STOP	-2.790073	0.028904	-96.5301	< 2.2e-16	***
Communication START	4.256786	0.040555	104.9624	< 2.2e-16	***
Communication STOP	-3.899224	0.040555	-96.1457	< 2.2e-16	***
Leisure START	5.509079	0.021683	254.0755	< 2.2e-16	***
Leisure STOP	-4.653257	0.021683	-214.6055	< 2.2e-16	***
Personal Care START	3.217249	0.068199	47.1741	< 2.2e-16	***
Personal Care STOP	-3.998140	0.068199	-58.6242	< 2.2e-16	***
NA START	2.698641	0.088388	30.5316	< 2.2e-16	***
NA STOP	-4.242211	0.088388	-47.9951	< 2.2e-16	***
Waiting START	1.343119	0.174078	7.7156	1.199e-14	***
Waiting STOP	-3.856553	0.174078	-22.1542	< 2.2e-16	***
Volunteering START	1.213907	0.185695	6.5371	6.273e-11	***
Volunteering STOP	-4.582844	0.185695	-24.6794	< 2.2e-16	***
Caregiving START	1.430130	0.166667	8.5808	< 2.2e-16	***
Caregiving STOP	-4.013357	0.166667	-24.0801	< 2.2e-16	***
Education START	-0.207479	0.377964	-0.5489	0.58305	
Education STOP	-4.302118	0.377964	-11.3823	< 2.2e-16	***
Work Production START	1.430130	0.166667	8.5808	< 2.2e-16	***
Work Production STOP	-4.863244	0.166667	-29.1795	< 2.2e-16	***
Inter.Sl.Ons	-0.890707	0.184920	-4.8167	1.459e-06	***
Inter.Sl.Dur	0.353266	0.187985	1.8792	0.06021	.
PerCare.Sl.Ons	1.314628	0.232737	5.6485	1.618e-08	***
PerCare.Sl.Dur	0.021551	0.243891	0.0884	0.92959	

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null deviance: 174603.4 on 18228 degrees of freedom

Residual deviance: 21489.30 on 18196 degrees of freedom

Chi-square: 153114.1 on 32 degrees of freedom, asymptotic p-value 0

AIC: 21553.30 AICC: 21553.41 BIC: 21803.24

Log posterior: -42776.04

Prior parameters: mu=0 sigma=1000 nu=4

We again note that, overall, sleep interruption is rare, the evidence for which is captured by the negative coefficient for `Inter.Sl.Ons` ($\beta = -0.891^{***}$), which, is interpreted as the conditional log hazard of returning to sleep versus doing some thing else; this has a correspondingly low

conditional probability of 0.0568. However, given that sleep interruption occurs at all, it tends to decrease the average sleeping spell by a factor of 0.7024 ($\frac{1}{e^{0.353266}}$), or roughly 90 minutes.

```
R> exp(-0.890707 + 4.989059) /
+   sum(exp(intfit.sleep$coef[grep("START", names(intfit.sleep$coef))]))

[1] 0.05677088
```

Moving onto interruptions directly caused by personal care needs, we see that, net of the overall tendency for interruption to occur, a spell of personal care that follows a sleeping spell increases the chances that the next event will indeed be sleeping. Ergo, personal care activities, such as using the bathroom, or changing clothes, interrupt spells of sleeping in this population (with conditional probability 0.5151). However, such interruptions do not significantly affect the average duration of sleep.

5.1. Dyadic data

As we have discussed throughout this paper, the general modeling function in the **relevent** package is `rem()`. For dyadic relational event data (where events are sender, receiver, action type tuples), the `rem.dyad()` function is appropriate for modeling large numbers of dyadic relational event statistics (currently, 34 classes of sufficient statistics are available, though users could specify more than that using the covariate terms). In principle, `rem()` can be used to fit models for dyadic relational event data as well as the ego-centric data demonstrated here; this allows support for a wider range of event types, self-directed events, support constraints etc., at the cost of the simplicity and computational optimizations afforded by `rem.dyad()`. While the primary purpose of the **informR** package is to aid in the construction of models for ego-centric data, some of its functions may be useful for constructing such dyadic models to use with `rem()`. As a proof-of-concept, a simple example of how **informR** tools can be used to construct dyadic models based on the example in the `rem.dyad()` documentation from the **relevent** package is provided in the Appendix A.

6. Conclusion

This paper has outlined the basic functionality of the **informR** package for R. Sufficient statistics to model event sequences for both ordinal and interval time data were constructed and employed in relational event models using real data and the **relevent** package. In addition, code for manipulating these statistics and constructing conditional models was introduced. The **informR** package greatly simplifies the use of the `rem()` modeling function, specifically, and the modeling of event histories with dependent sub-features more generally. It is hoped that this toolkit will lead to wider use of relational event models for complex event sequences, particularly by researchers engaged with non-dyadic data.

Computational details

The version of package **informR** used in this tutorial is 1.0-5 and the version of package **relevent** is 1.0-4.

The average batch run-time ($n = 30$) of the examples employed in this tutorial was 12.5 minutes on a 2.2 GHz Intel T6600 processor with 3.8 GB of RAM (absolute maximum memory consumption was 1.8 GB as estimated by the bourne-shell program `top`) using Linux kernel v2.6.32.

Acknowledgments

We would like to thank the anonymous reviewers for their useful comments and suggestions. Funding for the development of this project was supported by grants from the National Science Foundation (CMS-0624257) and the Office of Naval Research (N00014-08-1-1015), and revisions were supported by funding from the National Institutes of Health Intramural Research Program (Z01HG200335 to L. Koehly).

References

- Allignol A, Latouche A (2014). “CRAN Task View: Survival Analysis.” Version 2014-12-18, URL <http://CRAN.R-project.org/view=Survival>.
- Barker JC, Mitteness LS (1988). “Nocturia in the Elderly.” *The Gerontologist*, **28**(1), 99–104.
- Blossfeld H, Rohwer G (2002). *Techniques of Event History Analysis: New Approaches to Causal Analysis*. 2nd edition. Lawrence Erlbaum Associates.
- Brandes U, Lerner J, Snijders TAB (2009). “Networks Evolving Step by Step: Statistical Analysis of Dyadic Event Data.” In *Proceedings of the 2009 International Conference on Advances in Social Network Analysis and Mining (ASONAM 2009)*, pp. 200–205.
- Butts CT (2008). “A Relational Event Framework for Social Action.” *Sociological Methodology*, **38**(1), 155–200.
- Butts CT (2015). *relevent: Relational Event Models*. R package version 1.0-4, URL <http://CRAN.R-project.org/package=relevent>.
- DuBois C, Butts CT, McFarland D, Smyth P (2013). “Hierarchical Models for Relational Event Sequences.” *Journal of Mathematical Psychology*, **57**(6), 297–309.
- Friedl J (2006). *Mastering Regular Expressions*. O’Reilly Media, Inc, Sebastopol.
- Gentleman R (2010). *mu haz: Hazard Function Estimation in Survival Analysis*. R package version 1.2.5, URL <http://CRAN.R-project.org/package=muhaz>.
- Geyer CJ (2014). *trust: Trust Region Optimization*. R package version 0.1-6, URL <http://CRAN.R-project.org/package=trust>.
- Marcum CS (2015). *informR: R Tools for Creating Sequence Statistics*. R package version 1.0-5, URL <http://CRAN.R-project.org/package=informR>.
- Mills M (2011). *Introducing Survival and Event History Analysis*. Sage, Thousand Oaks.

- Plate T, Heiberger R (2011). *abind: Combine Multi-Dimensional Arrays*. R package version 1.4-0, URL <http://CRAN.R-project.org/package=abind>.
- R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <http://www.R-project.org/>.
- Therneau TM (2014). *survival: A Package for Survival Analysis in S*. R package version 2.37-7, URL <http://CRAN.R-project.org/package=survival>.
- Therneau TM, Grambsch PM (2000). *Modeling Survival Data: Extending the Cox Model*. Springer-Verlag, New York.
- Tikkinen KAO, Tammela T, Huhtala H, Auvinen A (2006). “Is Nocturia Equally Common Among Men and Women? A Population Based Study in Finland.” *The Journal of Urology*, **175**(2), 596–600.
- Willette-Murphy K, Todera C, Yeaworth R (2006). “Mental Health and Sleep of Older Wife Caregivers for Spouses with Alzheimer’s Disease and Unrelated Disorders.” *Issues in Mental Health Nursing*, **27**(8), 837–852.

A. Dyadic models

In this appendix, we demonstrate a proof-of-concept for fitting a model of dyadic data with `rem()` from the **relevent** package. Normally, dyadic data is modeled using `rem.dyad()` and the examples here are derived from the example code in that **relevent** package function. Since `rem.dyad()` and `rem()` are optimized using different routines – `optim()` and `trust` (Geyer 2014), respectively – results may differ slightly between models. The example is simple: we fit a relational event model of fixed-effects for sending and receiving. Extension to other sorts of dyadic relational event terms can be performed using techniques similar to those shown here.

In this example, we estimate fixed effects for sending and receiving ties among an artificial network of five actors using both estimating functions: `rem.dyad()` and `rem()`. With `rem.dyad()`, these terms are conveniently added to the dyadic relational events model by passing `c("FESnd", "FERec")` to the argument `effects` of that function. With `rem()`, however, the burden of generating the sufficient statistics to add to the model falls entirely on the user. Like the egocentric (non-dyadic) cases described in the main text of this manuscript, the **informR** tools are also helpful in this case. Specifically, we can use `gen.evl()` and `gen.intercepts()`, with minor modifications, to facilitate model fitting with `rem()`. For completeness, we fit both ordinal and interval likelihood models. Also, we demonstrate the use of maximum likelihood estimation (MLE) to fit these models.

We begin by drawing an artificial network from a simple relational events process unfolding among five actors:

```
R> set.seed(867)
R> n <- 5
R> tmax <- 25
R> roweff <- rnorm(n)
R> roweff <- roweff - roweff[1]
R> coleff <- rnorm(n)
R> coleff <- coleff - coleff[1]
R> lambda <- exp(outer(roweff, coleff, "+"))
R> diag(lambda) <- 0
R> ratesum <- sum(lambda)
R> esnd <- as.vector(row(lambda))
R> erec <- as.vector(col(lambda))
R> time <- 0
R> edgelist <- vector()
R> while (time < tmax) {
+   drawsr <- sample(1:(n^2), 1, prob = as.vector(lambda))
+   time <- time + rexp(1, ratesum)
+   if (time < tmax)
+     edgelist <- rbind(edgelist, c(time, esnd[drawsr], erec[drawsr]))
+   else
+     edgelist <- rbind(edgelist, c(tmax, NA, NA))
+ }
```

To fit a dyadic relational event model with `rem()`, we must convert the edgelist data used by

`rem.dyad()` to eventlist form. To do this, we first note that the set of all distinct event types in the sense of `rem()` consists of all ordered sender/receiver pairs, of which there are 20. We enumerate the set of event types here using simple sender/receiver concatenation, with the **informR** function `gen.evl()` thus producing an eventlist suitable for use with other **informR** routines. Also, since this relational events process occurs within a single observation period on a single network, the grouping factor column is set to an arbitrary constant value (here, 1). We also specify an exogenous event at the end of observation.

```
R> evl <- gen.evl(cbind(apply(edgelist[, 2:3], 1, paste, collapse = "."),
+   rep(1, NROW(edgelist))), null.events = "NA.NA")
```

The `evl` object now contains the required combination of timing and event type information. The latter can be found in the event key, i.e.:

```
R> evl$event.key
```

```
      id event.type
[1,] "a" "1.5"
[2,] "b" "4.3"
[3,] "c" "2.5"
[4,] "d" "5.3"
[5,] "e" "4.5"
[6,] "f" "4.2"
[7,] "g" "5.2"
[8,] "h" "2.3"
[9,] "i" "1.3"
[10,] "j" "4.1"
[11,] "k" "3.2"
[12,] "l" "2.1"
[13,] "m" "3.5"
[14,] "n" "5.1"
[15,] "o" "3.1"
[16,] "p" "5.4"
[17,] "q" "1.2"
[18,] "r" "1.4"
[19,] "s" "2.4"
[20,] "t" "NA.NA"
```

Note that the order of the event codes is based on the order in which events are encountered in the data, and hence the event key must be explicitly searched when creating statistics. Also, upon inspection of the event key, we note that this particular model did not exhaust all possible 20 ($n^2 - n$) events; actor 3 never sends to actor 4. One of the limits of the **informR** package is that it conditions on the observed set of actions and not on the set of possible actions. As a result, we need to append this potential event to the event key.

```
R> evl$event.key <- rbind(evl$event.key, c("u", "3.4"))
```


To fit the fixed effect model, we must create intercept dummies for each event involving the same sender, and as well as a set for each receiver. We can accomplish this by first using `gen.intercepts()` with the `contrasts` parameter set to `FALSE` (`gen.intercepts(evl, contr = FALSE)`) and then manipulating the resulting `statslist` array directly.

```
R> ev.ints <- gen.intercepts(evl, contr = FALSE)
R> sformlist <- c(
+   lapply(2:n, function(z) {
+     str <- paste(z, ".", sep = "")
+     grep(str, evl$event.key[-20, 2])}),
+   lapply(2:n, function(z) {
+     str <- paste(".", z, sep = "")
+     grep(str, evl$event.key[-20, 2])
+   })
+ )
R> b1 <- lapply(sformlist, function(x) ev.ints[[1]][[1]][, , x])
R> b1.1 <- lapply(b1, apply, MARGIN = 1:2, sum)
R> FEs <- array(unlist(b1.1), dim = c(nrow(b1.1[[1]]), ncol(b1.1[[1]]),
+   length(b1.1)))
R> dimnames(FEs) <- list(dimnames(b1[[1]])[[1]], dimnames(b1[[1]])[[2]],
+   c(paste("FESnd", 2:n, sep = "."), paste("FERec", 2:n, sep = ".")))
```

We can then fit the models using `rem.dyad()` and `rem()`. There are minor differences in function signatures and default behavior as the two functions have different development histories. Here, we have set the argument `fit.method` in `rem.dyad()` to `"MLE"`, which is equivalent to the argument `estimator` in `rem()` and we set the argument `hessian` to `TRUE` in `rem.dyad()`.

```
R> fit.rem.dyad <- rem.dyad(edgelist, n, effects = c("FESnd", "FERec"),
+   fit.method = "MLE", hessian = TRUE)
R> fit.rem <- rem(evl$eventlist, sfl2statslist(list(FEs)), estimator = "MLE")
R> lapply(list(rem.dyad = fit.rem.dyad, rem = fit.rem), summary)
```

```
$rem.dyad
```

```
Relational Event Model (Ordinal Likelihood)
```

	Estimate	Std.Err	Z value	Pr(> z)
FESnd.2	0.115533	0.075571	1.5288	0.1263159
FESnd.3	-1.427850	0.137976	-10.3486	< 2.2e-16 ***
FESnd.4	0.581748	0.067459	8.6237	< 2.2e-16 ***
FESnd.5	-0.345661	0.101777	-3.3962	0.0006832 ***
FERec.2	-0.258956	0.112914	-2.2934	0.0218250 *
FERec.3	0.893957	0.085375	10.4709	< 2.2e-16 ***
FERec.4	-1.927157	0.226697	-8.5010	< 2.2e-16 ***
FERec.5	1.253683	0.084184	14.8922	< 2.2e-16 ***

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```

Null deviance: 4871.061 on 813 degrees of freedom
Residual deviance: 4074.199 on 805 degrees of freedom
      Chi-square: 796.862 on 8 degrees of freedom, asymptotic p-value 0
AIC: 4090.199 AICC: 4090.378 BIC: 4127.804

```

```
$rem
```

```
Egocentric Relational Event Model (Ordinal Likelihood)
```

	MLE	Std.Err	Z value	Pr(> z)
FESnd.2	0.115526	0.075571	1.5287	0.1263399
FESnd.3	-1.427881	0.137977	-10.3487	< 2.2e-16 ***
FESnd.4	0.581745	0.067459	8.6236	< 2.2e-16 ***
FESnd.5	-0.345653	0.101777	-3.3962	0.0006833 ***
FERec.2	-0.258950	0.112913	-2.2934	0.0218271 *
FERec.3	0.893943	0.085375	10.4708	< 2.2e-16 ***
FERec.4	-1.927213	0.226701	-8.5011	< 2.2e-16 ***
FERec.5	1.253660	0.084183	14.8920	< 2.2e-16 ***

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Null deviance: 4871.061 on 813 degrees of freedom
```

```
Residual deviance: 4074.199 on 805 degrees of freedom
```

```
      Chi-square: 796.862 on 8 degrees of freedom, asymptotic p-value 0
```

```
AIC: 4090.199 AICC: 4090.378 BIC: 4127.804
```

As can be seen by the output, the results of the two ordinal model fitting routines are quite consistent. As we discussed above, there are minor differences in precision between the output of `rem.dyad()` and `rem()`. These small discrepancies arise due to the fact that the two functions use different optimizer routines: `optim()`, and `trust()`, respectively.

Finally, for completeness, we fit the interval timing likelihood models thusly:

```

R> evl2 <- gen.evl(cbind(apply(edgelist[, 2:3], 1, paste, collapse = "."),
+   edgelist[, 1], rep(1, NROW(edgelist))), null.events = "NA.NA")
R> evl2$event.key <- rbind(evl2$event.key, c("u", "3.4"))
R> fit.rem.dyad2 <- rem.dyad(edgelist, n, effects = c("FESnd", "FERec"),
+   fit.method = "MLE", ordinal = FALSE, hessian = TRUE)
R> fit.rem2 <- rem(evl2$eventlist, sfl2statslist(list(FES)),
+   estimator = "MLE", timing = "interval")
R> lapply(list(rem.dyad = fit.rem.dyad2, rem = fit.rem2), summary)

```

As with the ordinal cases, these models are consistent and, happily, lack the null-deviance bug:

```
$rem.dyad
```

```
Relational Event Model (Temporal Likelihood)
```

	Estimate	Std.Err	Z value	Pr(> z)
FESnd.2	0.109054	0.063964	1.7049	0.088207 .

```

FESnd.3 -1.434714  0.131277 -10.9289 < 2.2e-16 ***
FESnd.4  0.575397  0.054698  10.5196 < 2.2e-16 ***
FESnd.5 -0.353124  0.090571  -3.8989 9.665e-05 ***
FERec.2 -0.269518  0.091848  -2.9344  0.003342 **
FERec.3  0.883657  0.056590  15.6152 < 2.2e-16 ***
FERec.4 -1.938053  0.216464  -8.9532 < 2.2e-16 ***
FERec.5  1.243079  0.052571  23.6457 < 2.2e-16 ***

```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null deviance: 835.564 on 813 degrees of freedom

Residual deviance: 38.71497 on 806 degrees of freedom

Chi-square: 796.849 on 7 degrees of freedom, asymptotic p-value 0

AIC: 54.71497 AICC: 54.89408 BIC: 92.32082

\$rem

Egocentric Relational Event Model (Interval Likelihood)

	MLE	Std.Err	Z value	Pr(> z)
FESnd.2	0.109054	0.063964	1.7049	0.088207 .
FESnd.3	-1.434712	0.131277	-10.9289	< 2.2e-16 ***
FESnd.4	0.575397	0.054698	10.5196	< 2.2e-16 ***
FESnd.5	-0.353124	0.090571	-3.8989	9.665e-05 ***
FERec.2	-0.269519	0.091848	-2.9344	0.003342 **
FERec.3	0.883657	0.056590	15.6152	< 2.2e-16 ***
FERec.4	-1.938054	0.216464	-8.9532	< 2.2e-16 ***
FERec.5	1.243079	0.052571	23.6457	< 2.2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null deviance: 835.564 on 813 degrees of freedom

Residual deviance: 38.71497 on 805 degrees of freedom

Chi-square: 796.849 on 8 degrees of freedom, asymptotic p-value 0

AIC: 54.71497 AICC: 54.89408 BIC: 92.32082

Affiliation:

Christopher Steven Marcum
National Human Genome Research Institute
National Institutes of Health
Bethesda, MD, United States of America
E-mail: chris.marcum@nih.gov

Carter T. Butts
Departments of Sociology, Statistics, and EECS, and
Institute for Mathematical Behavioral Sciences
University of California, Irvine
Irvine, CA, United States of America
E-mail: buttsc@uci.edu