What's for dynr: A Package for Linear and Nonlinear Dynamic Modeling in R

Lu Ou[+]
Pennsylvania State University

Michael D. Hunter[+]
University of Oklahoma Health Sciences Center

Sy-Miin Chow
Pennsylvania State University

[+]These two authors contributed equally to the work. This paper is currently under review.

Abstract

Intensive longitudinal data have become increasingly prevalent in studies of circadian rhythms, emotions, propagation of diseases, addictive behaviors, dyadic and family-level interactions, as well as other human dynamic processes. Such data are noisy, often multivariate in nature, and may involve multiple subjects undergoing regime switches (e.g., showing discontinuities interspersed with continuous dynamics). Despite increasing interest in using differential/difference equation models for representing these processes, there has been a scarcity of software packages that are fast, freely accessible, and amenable to the modeling goals of researchers of human dynamics. We have created an R package that is based on novel and computationally efficient algorithms for handling a broad class of linear and nonlinear discrete- and continuous-time models with regime-switching properties and linear Gaussian measurement functions in C, while maintaining simple and easy-to-learn model specification functions in R. We present the mathematical and computational basis used by the **dynr** R package, and present two illustrative examples to demonstrate the unique features of **dynr**.

Keywords: dynamic modeling, regime switching, nonlinear, factor analysis, Markov model, state-space model.

1. Introduction

The past several decades have seen a significant rise in the use of intensive longitudinal designs, particularly in the social and behavioral sciences. These designs – implemented in the form of daily diary studies, ecological momentary assessments, ambulatory assessments and other related variations – aim to capture change processes "in the moment" as they unfold within subjects (Bolger and Laurenceau 2013). The intensive longitudinal data (ILD) that result from such designs have permeated our everyday lives in more than one way. For instance, clinical trials now routinely incorporate electronic patient-reported outcomes where

individuals provide periodic reports from home as part of their daily routines (Byrom and Tiplady 2010). These ILD provide valuable information about the effectiveness, side-effects, and optimal timing of patient care (Stone, Shiffman, Atienza, and Nebeling 2008). Similarly, mobile devices and wearable sensors are facilitating ambulatory collection of data that were previously confined to laboratory settings (e.g., heart rate and skin conductance; Schnell, Maini, Newman, and Newman 2008). However, the richness and sheer quantity of ILD that are now being collected also lead to the emergence of novel data analytic challenges, and a dire need for methods that can better manage and handle the dynamics of ILD.

Differential equation models and difference equation models (e.g., in the form of state-space models) have been one of the most dominant tools for representing the dynamics of ILD in disciplines such as the physical sciences, econometrics, engineering, and ecology. In parallel, some computational advances have been proposed in estimating *regime-switching* models – namely, models positing how otherwise continuous dynamic processes may undergo discontinuous changes through categorical but unobserved phases known as "regimes" (Kim and Nelson 1999; Hamilton 1989; Muthén and Asparouhov 2011; Chow, Grimm, Guillaume, Dolan, and McArdle 2013; Chow, Witkiewitz, Grasman, and Maisto 2015; Dolan 2009). One of the best known examples from psychology is perhaps Piaget's (1969) theory of human cognitive development and related extensions (Dolan, Jansen, and Van der Maas 2004; van der Maas and Molenaar 1992; Hosenfeld 1997). Other examples include Kohlberg's (Kohlberg and Kramer 1969) conceptualization of stagewise development in moral reasoning, Van Dijk and Van Geert's (2007) findings on discrete shifts in early language development, as well as Fukuda and Ishihara's (1997) work on the discontinuous changes in infant sleep and wakefulness rhythm during the first six months of life. In previous work in the literature, the timing and nature of the switches between regimes have been specified as a first- and higher-order Markovian process (e.g., Hamilton 1989; Dolan 2009; Yang and Chow 2010; Chow and Zhang 2013; Chow *et al.* 2015); or as governed by deterministic thresholds (e.g., as in threshold autoregressive models; Tong and Lim 1980), past values of a system (e.g., as in self-exciting threshold autoregressive models; Tiao and Tsay 1994), and external covariates of interest (Muthén and Asparouhov 2011; Chow *et al.* 2013, 2015).

Several programs and packages exist for fitting differential equation and difference equation models. However, each program has certain limitations that **dynr** aims to overcome. Speaking broadly, the largest differences between **dynr** and other packages are threefold: (1) **dynr** readily allows for multi-subject models, (2) **dynr** allows for nonlinear dynamics, and (3) **dynr** allows for regime switching throughout every part of the model. Many R packages exist for univariate and multivariate time series. CRAN lists 217 packages in its task view for time series (Hyndman 2016), a complete review of which is well-beyond the scope of this work. However, generally these packages lack facilities for fitting time series from multiple subjects (see Table 3.2 for an overview)[1]. Likewise there are very few software utilities designed for nonlinear dynamics or regime switching. Petris and Petrone (2011) reviewed three packages for linear state-space models: **dlm** (Petris 2010), **KFAS** (Helske 2016), and **dse** (Gilbert 2006 or later). These are among the state of the art for state-space modeling in R. Although **KFAS** can accommodate in its measurement model all densities within the exponential family, the corresponding dynamic model is required to be linear. In addition to these R packages, the **OpenMx** 2.0 release (Neale, Hunter, Pritikin, Zahery, Brick, Kirkpatrick, Estabrook, Bates,

[1]The standard work around for packages that allow multivariate time series is to treat each subject as a new set of variables in the time series. This becomes untenable with a large number of subjects.

Maes, and Boker 2016) has maximum likelihood time-varying linear discrete- and continuous-time state-space modeling. Likewise, the **MKFM6** program (Dolan 2005) implements methods of Harvey (1989) for time-invariant linear state-space models. **SsfPack** (Koopman, Shephard, and Doornik 1999) implements the methods of Durbin and Koopman (2001) for linear state-space modeling and Markov chain Monte Carlo methods for nonlinear modeling, but it is primarily restricted to single-subject time series without regime switching. The **ctsem** package (Driver, Oud, and Voelkle in press) has utilities for linear state-space modeling of multiple subjects in continuous time, but lacks functionality for nonlinear models or regime switching. MATLAB (The MathWorks, Inc. 2016) has numerous extensions for time series and state-space modeling (e.g., Grewal and Andrews 2008), but lacks the ability to include regime switching and multiple subjects. Helske (2016) included a review of numerous other packages for non-Gaussian time series models which generally do not involve latent (unobserved) variables.

Overall, developments in fitting differential/difference models that evidence discontinuities in dynamics are still nascent. Despite some of the above-mentioned advances in computational algorithms, there is currently no readily available and accessible software package that allows researchers to fit differential/difference equations with regime-switching properties. As stated previously, currently available computational programs for dynamic modeling are limited in one of several ways: (1) they are restricted to handling only linear differential or difference equation models within regimes such as the package **OpenMx**; (2) they can only handle very specific forms of nonlinear relations among latent variables; (3) they are computationally slow; (4) they do not allow for stochastic qualitative shifts in the dynamics over time (i.e., regime switching); or (5) they require that the user write complex compiled code to enhance computational speed at the cost of high user burden. Efficient and user-friendly computer software needs to be developed to overcome these restrictions so the estimation of dynamic models can become more applicable and accessible by researchers.

We present an R (R Core Team 2015) package, **dynr**, that allows users to fit both linear and nonlinear differential and difference equation models with regime-switching properties. All computations are performed quickly and efficiently in C, but are tied to a user interface in the familiar R language. Specifically, for a very broad class of linear and nonlinear differential/difference equation models with linear Gaussian measurement functions, **dynr** provides R helper functions that write and compile the C functions based on user input in R so that the user never has to write or even see the C code that underlies **dynr**. This removes some of the barriers to dynamic modeling, opening it as a possibility to a broader class of users, while retaining the flexibility of specifying targeted model-specific functions in C for users wishing to pursue models that are not yet supported in the R interface.

In the remaining sections, we will first present the mathematical and computational bases of the **dynr** R package, and then demonstrate the interface of **dynr** for modeling multivariate observations with Gaussian measurement errors using two ILD modeling examples from the social and behavioral sciences. Key features of the **dynr** package we seek to highlight include: (1) **dynr** fits discrete- and continuous-time dynamic models to multivariate longitudinal/time-series data; (2) **dynr** deals with dynamic models with regime-switching properties; (3) for improved speed, **dynr** computes and optimizes negative log-likelihood function values in C; (4) **dynr** handles linear and nonlinear dynamic models with an easy-to-use interface that includes a matrix form (for linear dynamic models only) and formula form (for linear as well as nonlinear models); (5) **dynr** removes the burden on the user to perform analytic differentiation in fitting nonlinear differential/difference equation models by providing the

user with R's automatic differentiation; and (6) **dynr** provides ready-to-present results through LaTeX equations and plots.

2. General modeling framework

In this section, we discuss the broader modeling framework of **dynr**, of which the models shown in the illustrative examples can be viewed as special cases. At the very basic level, our general modeling framework comprises a dynamic model and a measurement model. The former describes the ways in which the latent variables change over time whereas the latter portrays the relationships between a set of observed variables and a set of latent variables at a specific time. In cases involving multiple-regime models, the general model also comprises two multinomial logistic regression models governing, respectively, the initial regime probabilities and transition probabilities between regimes. Both the dynamic and measurement models may show regime-dependent properties.

The dynamic model can take on the form of continuous-time or discrete-time functions. More formally, for continuous-time models, we assume that the dynamic model within a particular regime takes on the form of

$$d\boldsymbol{\eta}_i(t) = \boldsymbol{f}_{S_i(t)}\left(\boldsymbol{\eta}_i(t), t, \boldsymbol{x}_i(t)\right) dt + d\boldsymbol{w}_i(t), \tag{1}$$

where i indexes person, t indexes time, $\boldsymbol{\eta}_i(t)$ is the $r \times 1$ vector of latent variables at time t, $\boldsymbol{x}_i(t)$ is the vector of covariates at time t, and $\boldsymbol{f}_{S_i(t)}(.)$ is the vector of (possibly nonlinear) dynamic functions. Note that $\boldsymbol{f}_{S_i(t)}(.)$ depends on the latent regime indicator, $S_i(t)$, the discrete-valued latent variable that indexes the operating regime at time t. Throughout, we use the term *regime* and *class* interchangeably. The left-hand side of Equation 1, $d\boldsymbol{\eta}_i(t)$, gives the differential of the vector of continuous latent variables, $\boldsymbol{\eta}_i(t)$. Thus, changes in the vector of latent variables are specified as (possibly nonlinear) functions of the same set of latent variables, external covariates, and time. These functions, shown as $\boldsymbol{f}_{S_i(t)}(.)$, are often known as the *drift* functions in the stochastic differential equation literature. Added to these deterministic changes induced by the drift functions is $\boldsymbol{w}_i(t)$, an r-dimensional Wiener process (i.e., continuous-time analog of a random walk process). The differentials of the Wiener processes have zero means and regime-specific covariance matrix, $\mathbf{Q}_{S_i(t)}$, often called the *diffusion* matrix.

For discrete-time processes, we adopt a dynamic model in state-space form as (Durbin and Koopman 2001)

$$\boldsymbol{\eta_i}(t_{i,j+1}) = \boldsymbol{f}_{S_i(t)}\left(\boldsymbol{\eta_i}(t_{i,j}), t_{i,j}, \boldsymbol{x}_i(t_{i,j})\right) + \boldsymbol{w}_i(t_{i,j+1}), \tag{2}$$

now postulated to unfold at discrete time points indexed by sequential positive integer values of t. This is a one-step-ahead or difference equation form that is typically seen in discrete-time state-space models. In this case, $\boldsymbol{w}_i(t)$ denotes a vector of Gaussian distributed process noise with regime-specific covariance matrix, $\mathbf{Q}_{S_i(t)}$. Note that even though continuous- and discrete-time models are typically expressed in the literature with distinct notations with respect to time, here we have intentionally kept our notation largely similar between the two sets of models to facilitate the linkage to software code, which allows for easy and seamless transition between the discrete- and continuous-time frameworks with the use of a simple binary flag. In a similar vein, we refer to $\boldsymbol{f}_{S_i(t)}(.)$ in both Equations 1 and 2 broadly as the

dynamic functions. We acknowledge that in the broader literature, $\boldsymbol{f}_{S_i(t)}(.)$ have been called the continuous-time dynamic functions, the vector field of the differential equations, or the drift functions in the continuous-time case; whereas in the discrete-time case, $\boldsymbol{f}_{S_i(t)}(.)$ have been called the discrete-time dynamic functions, the flow of the differential equations, or the state-transition functions.

In both the discrete- and continuous-time cases, the initial conditions for the dynamic functions are defined explicitly to be the latent variables at an individual-specific initial time point, $t_{i,1}$ (i.e., the first observed time point), denoted as $\boldsymbol{\eta}_i(t_{i,1})$, and are specified to be normally distributed with means $\boldsymbol{\mu}_{\boldsymbol{\eta}_1}$ and covariance matrix, $\boldsymbol{\Sigma}_{\boldsymbol{\eta}_1}$:

$$\boldsymbol{\eta}_i(t_{i,1}) \sim N\left(\boldsymbol{\mu}_{\boldsymbol{\eta}_1}, \boldsymbol{\Sigma}_{\boldsymbol{\eta}_1}\right). \tag{3}$$

Likewise for both discrete- and continuous-time models, we assume that the latent variables are measured only at discrete times. That is, the latent variables may exist in discrete or continuous time, but observations only occur at selected, discrete time points. Thus, we have a discrete-time measurement model in which $\boldsymbol{\eta}_i(t_{i,j})$ at discrete time point $t_{i,j}$ is indicated by a $p \times 1$ vector of manifest observations, $\boldsymbol{y}_i(t_{i,j})$. Generally for continuous-time processes, the time intervals, $\Delta_{i,j} = t_{i,j+1} - t_{i,j}$ may be uneven, whereas in discrete-time processes, they must be uniform. However, missing data may be present under either of these specifications, and this is one way of handling unequal measurement intervals in discrete time. The vector of manifest observations is linked to the latent variables as

$$\begin{aligned} \boldsymbol{y}_i(t_{i,j}) = \boldsymbol{\tau}_{S_i(t_{i,j})} + \boldsymbol{\Lambda}_{S_i(t_{i,j})}\boldsymbol{\eta}_i(t_{i,j}) + \mathbf{A}_{S_i(t_{i,j})}\boldsymbol{x}_i(t_{i,j}) + \boldsymbol{\epsilon}_i(t_{i,j}), \\ \boldsymbol{\epsilon}_i(t_{i,j}) \sim N\left(\mathbf{0}, \mathbf{R}_{S_i(t_{i,j})}\right), \end{aligned} \tag{4}$$

where $\boldsymbol{\tau}_{S_i(t_{i,j})}$ is a $p \times 1$ vector of intercepts, $\mathbf{A}_{S_i(t_{i,j})}$ is a matrix of regression weights for the covariates observed at time $t_{i,j}$, $\boldsymbol{\Lambda}_{S_i(t_{i,j})}$ is a $p \times r$ factor loadings matrix that links the observed variables to the latent variables, and $\boldsymbol{\epsilon}_i(t_{i,j})$ is a $p \times 1$ vector of measurement errors assumed to be serially uncorrelated over time and normally distributed with zero means and (possibly) regime-specific covariance matrix, $\mathbf{R}_{S_i(t_{i,j})}$. Of course, all parts of the measurement model may be regime-dependent.

The subscript $S_i(t)$ that appears in Equations 1–4 indicates that the values of the parameters in these functions and matrices may depend on $S_i(t)$, the operating regime for individual i at time point, t. Often in practice, only some of these elements are freed to vary by regime. To make inferences on $S_i(t_{i,j})$, it is essential to specify a model or mechanism through which $S_i(t_{i,j})$ changes over individuals and time. Just as with the continuous latent variables in $\boldsymbol{\eta}_i(t)$, we initialize the categorical latent variable $S_i(t_{i,j})$ on the first occasion and then provide a model for how $S_i(t_{i,j})$ changes over time. The initial class (or regime) probabilities for $S_i(t_{i,1})$ are represented using a multinomial regression model as

$$\Pr\left(S_i(t_{i,1}) = m | \boldsymbol{x}_i(t_{i,1})\right) \quad \stackrel{\Delta}{=} \pi_{m,i1} = \frac{\exp(a_m + \mathbf{b}_m^T \boldsymbol{x}_i(t_{i,1}))}{\sum_{k=1}^{M} \exp(a_k + \mathbf{b}_k^T \boldsymbol{x}_i(t_{i,1}))}, \tag{5}$$

where M denotes the total number of regimes, a_m denotes the logit intercept for the mth regime and $\mathbf{b}_m$ is a $n_b \times 1$ vector of regression slopes linked to a vector of covariates used to explain possible interindividual differences in initial log-odds (LO) of being in a regime relative to the reference regime selected by the user, operationalized as the regime where a_m and all

entries in $\mathbf{b}_m$ are set to zero. Setting these entries to be zero in at least the reference regime is necessary for identification purposes: this ensures that the initial regime probabilities across all the hypothesized regimes sum to 1.0. In the simplest case without covariates, Equation 5 reduces to a specification of M initial regime prevalence parameters - either on a LO (ranging from $-\infty$ to $+\infty$) or a probability (ranging between 0 and 1) scale, as preferred by the user.

With the initial class probabilities specified, it remains to create a model for how the classes change over time. A simple strategy for this is to create a first-order Markov model for the categorical latent variable, $S_i(t_{i,j})$, $j = 2, \ldots, T$, for the remaining time span. Such a model assumes that the probability of entering the current regime depends only on the previous regime. All possible transitions from one regime to another can be arranged into a matrix of transition probabilities, in which the rows index the previous regime at time $t_{i,j-1}$ and the columns index the regime to which the system transitions at time $t_{i,j}$. Hence, we use a first-order Markov process to define how the classes change over time in a transition probability matrix. This transition matrix may also depend on covariates. Thus, a multinomial logistic regression equation is assumed to govern the probabilities of transitions between regimes as:

$$\Pr\left(S_i(t_{i,j}) = m | S_i(t_{i,j-1}) = l, \boldsymbol{x}_i(t_{i,j})\right) \quad \triangleq \pi_{lm,it} = \frac{\exp(c_{lm} + \mathbf{d}_{lm}^T \boldsymbol{x}_i(t_{i,j})}{\sum_{k=1}^{M} \exp(c_{lk} + \mathbf{d}_{lk}^T \boldsymbol{x}_i(t_{i,j}))}, \tag{6}$$

where $\pi_{lm,it}$ denotes individual i's probability of transitioning from class l at time $t_{i,j-1}$ to class m at time $t_{i,j}$ (i.e., the entry in the lth row and mth column of the transition probability matrix), c_{lm} denotes the logit intercept for the transition probability, and $\mathbf{d}_{lm}$ is a $n_d \times 1$ vector of logit slopes summarizing the effects of the covariates in $\boldsymbol{x}_i(t_{i,j})$ on that transition probability. The coefficients in $\mathbf{d}_{lm}$ are LO parameters representing the effects of the covariates on the LO of transitioning from the lth regime into the mth regime relative to transitioning into the reference regime - namely, the regime in which all LO parameters (including c_{lM} and all elements in $\mathbf{d}_{lM}^T$) are set to 0. One regime, again, has to be specified as the reference regime for identification purposes to ensure that conditional on being in a particular regime at time $t_{i,j-1}$, the probabilities of transitioning to each of the M regimes sum to 1.0 (i.e., $\sum_{m=1}^{M} \pi_{lm} = 1$).

To summarize, the model depicted in Equations 1 – 6 may take on the form of various linear or nonlinear dynamic models in continuous- or discrete-time. Moreover, these dynamic models may have regime-switching properties. Systematic between-person differences stem primarily from changes in the person- and time-specific regime, $S_i(t_{i,j})$, and the corresponding changes in the dynamic and measurement models over persons and over occasions. In other words, by allowing the value of $S_i(t_{i,j})$ to vary over persons and time points, our general modeling framework has the capacity to yield heterogeneous and distinct-looking model-implied trajectories for different individuals. However, we assume that there are principled or nomothetic (group-based) laws in how individuals change over time within each regime. Thus, the hypothesized model is generally a group-based model that confines all modeling parameters to be invariant across multiple subjects.[2]

[2]There is, however, the possibility of incorporating additional sources of between-person differences in the form of random effects as additional latent variables in the model. We did not provide explicit illustrative examples to demonstrate this modeling variation.

3. Estimation procedures

In this section, we outline the procedures implemented in **dynr** for estimating the model shown in Equations 1 – 6. Because our focus is on presenting the **dynr** interface functions in R and the estimation algorithms and the corresponding Monte Carlo studies for testing their performance have been reported elsewhere, we only provide a brief outline of the key estimation procedures here and refer the reader to elsewhere for further technical details. We will begin with the procedures for handling discrete-time models due to their longer history and relative familiarity to readers across multiple disciplines, followed by the adaptations implemented to handle continuous-time models at the dynamic level. An overview of the estimation procedures involved, the different special cases handled by **dynr**, and the software packages that can handle these special cases are summarized in Table 3.2.

3.1. Discrete-time models

The estimation procedures implemented in **dynr** are designed to handle the various special cases subsumed under our broader modeling framework (see Table 3.2). Broadly speaking, these methods are based on the Kalman filter (Kalman 1960), its various continuous-time and nonlinear extensions, and the Kim filter (Anderson and Moore 1979; Bar-Shalom, Li, and Kirubarajan 2001; Kim and Nelson 1999; Yang and Chow 2010; Chow and Zhang 2013; Kulikov and Kulikova 2014; Kulikova and Kulikov 2014; Chow, Ou, Ciptadi, Prince, Rehg, Rozga, and Messinger Under review). A full and detailed explanation of these algorithms is beyond the scope of this paper; only a brief summary can be provided here. Readers may find Chow, Ho, Hamaker, and Dolan (2010) and the applicable sections of Neale *et al.* (2016) helpful in understanding the Kalman filter and its role in estimating the latent variables in $\boldsymbol{\eta}_i$. The Kim filter, designed to extend the Kalman filter to handle regime-switching state-space models, was proposed by Kim and Nelson (1999) and extended by Chow and Zhang (2013) to allow for nonlinear dynamic functions. Full details are given in the citations used in the rest of this section. By combining the Kalman filter, the Kim filter, and their extensions to continuous-time and nonlinear dynamics, a very large class of useful models can be estimated. In **dynr**, models are allowed to (1) be in discrete or continuous time, (2) be single-regime or regime-switching, (3) have linear or nonlinear dynamics, (4) involve stochastic or deterministic dynamics, and (5) have one or more subjects. All combinations of these variations are possible in **dynr**, creating 32 different kinds of models.

Let all parameters that appear in Equations 1 – 6 be collected into a vector of parameters, $\boldsymbol{\theta}$. In cases involving linear dynamics in discrete time without regime-switching properties, the model reduces to a standard linear state-space model, and we apply the Kalman filter (KF; Kalman 1960) to estimate the latent variable values and obtain other by-products for parameter optimization purposes. At each time point, the KF consists of two steps. In the first step, the dynamics are used to make a prediction for the latent state at the next time point conditional on the observed measurements up to time $t_{i,j-1}$, creating a predicted mean $\hat{\boldsymbol{\eta}}_i(t_{i,j}|t_{i,j-1}) = E(\boldsymbol{\eta}_i(t_{i,j})|\mathbf{Y}_i(t_{i,j-1}))$ and covariance matrix for the latent state $\mathbf{P}_i(t_{i,j}|t_{i,j-1}) = Cov[\boldsymbol{\eta}_i(t_{i,j})|\mathbf{Y}_i(t_{i,j-1})]$, where $\mathbf{Y}_i(t_{i,j-1})$ includes manifest observations at time $t_{i,1}$, $t_{i,2}$, ., up to time $t_{i,j-1}$. In the second step, the prediction is updated based on the measurement model (Equation 4) and the actual measurements from the next time point, yielding $\hat{\boldsymbol{\eta}}_i(t_{i,j}|t_{i,j}) = E(\boldsymbol{\eta}_i(t_{i,j})|\mathbf{Y}_i(t_{i,j}))$ and associated covariance matrix, $\mathbf{P}_i(t_{i,j}|t_{i,j}) = Cov[\boldsymbol{\eta}_{it}|\mathbf{Y}_i(t_{i,j})]$. These predictions and errors are by-products of the KF and can be used

to construct a log-likelihood function known as the *prediction error decomposition* function (De Jong 1988; Harvey 1989; Hamilton 1994; Chow *et al.* 2010). This log-likelihood function is constructed based on the premise that the prediction errors, defined as $\mathbf{Y}_i(t_{i,j}) - E(\mathbf{Y}_i(t_{i,j})|\mathbf{Y}_i(t_{i,j}))$, which captures the discrepancies between the manifest observations and the predictions implied by the model, are multivariate normally distributed. This log-likelihood function is optimized to yield maximum-likelihood estimates of all the time-invariant parameters in the free parameter vector $\boldsymbol{\theta}$, as well as to construct information criterion (IC) measures (Chow and Zhang 2013; Harvey 1989) such as the Akaike Information Criterion (AIC; Akaike 1973) and Bayesian Information Criterion (BIC; Schwarz 1978). Standard errors of the parameter estimates are obtained by taking the square root of the diagonal elements of the inverse of the negative numerical Hessian matrix of the prediction error decomposition function at the point of convergence.

At convergence, other products from the linear KF include updated latent states, $\hat{\boldsymbol{\eta}}_i(t_{i,j}|t_{i,j})$, and the updated latent covariance matrices, $\mathbf{P}_i(t_{i,j}|t_{i,j})$. In the social and behavioral sciences, the entire time series of observations has often been collected prior to model fitting. In such cases, we use the fixed interval smoother (Anderson and Moore 1979; Ansley and Kohn 1985) to refine the latent variable estimates, yielding the smoothed latent variable estimates, $\hat{\boldsymbol{\eta}}_i(t_{i,j}|T_i) = E(\boldsymbol{\eta}_i(t_{i,j})|\mathbf{Y}_i(T_i))$, and associated covariance matrices, $\mathbf{P}_i(t_{i,j}|T_i)$.

When the dynamic model takes on the form of a nonlinear state-space model with differentiable dynamic functions, the linear KF is replaced with the extended Kalman filter (EKF; Anderson and Moore 1979; Bar-Shalom *et al.* 2001) so that the nonlinear dynamic functions are "linearized" or approximated by Taylor series expansion retaining only the first-order terms. Assuming that the measurement and process noise components are normally distributed and that the measurement equation is linear as assumed in Equation 4, the prediction errors are still multivariate normally distributed. These prediction errors can then be used to construct a log-likelihood function similar in form to the prediction error decomposition function as in the linear state-space modeling case, but the corresponding parameter estimates are only "approximate" ML estimates due to the truncation errors from the first-order Taylor series expansion in the EKF. The feasibility of this approach has been demonstrated by Chow, Ferrer, and Nesselroade (2007).

When a linear state-space model is used as the dynamic model but it is characterized by regime-switching properties, **dynr** uses an extension of the standard linear KF procedure, known as the Kim filter, and the related Kim smoother (Kim and Nelson 1999; Yang and Chow 2010), for estimation purposes. The Kim filter combines the linear KF, the Hamilton filter (Hamilton 1989) and a collapsing procedure to avoid the need to store M^2 new values of $\hat{\boldsymbol{\eta}}_i(t_{i,j}|t_{i,j})^{l,m} \triangleq \mathsf{E}[\boldsymbol{\eta}_i(t_{i,j})|S_i(t_{i,j-1}) = l, S_i(t_{i,j}) = m, \mathbf{Y}_i(t_{i,j})]$, as well as $\mathbf{P}_i(t_{i,j}|t_{i,j})^{l,m} \triangleq \mathrm{Cov}[\boldsymbol{\eta}_i(t_{i,j})|S_i(t_{i,j-1}) = l, S_i(t_{i,j}) = m, \mathbf{Y}_i(t_{i,j})]$ with each additional time point. The collapsing procedure averages the estimates over the previous regime l ($l = 1, \ldots, M$) so only the marginal estimates, $\hat{\boldsymbol{\eta}}_i(t_{i,j}|t_{i,j})^m$ (i.e., $E[\boldsymbol{\eta}_i(t_{i,j})|S_i(t_{i,j}) = m, \mathbf{Y}_i(t_{i,j})]$), and the associated covariance matrix, $\mathbf{P}_i(t_{i,j}|t_{i,j})^m$, $m = 1, \ldots, M$) need to be stored at each time step. To handle cases in which nonlinearities are present in Equation 2, a method proposed by Chow and Zhang (2013), called the extended Kim filter, is used for estimation instead. The extended Kim filter replaces the linear KF portion of the Kim filter with the nonlinear EKF.

3.2. Continuous-time models

Finally, when the dynamic model takes on the form of a continuous-time model – whether as composed of linear or nonlinear dynamic functions – the resultant estimation procedures are the continuous-discrete extended Kalman filter (CDEKF; Bar-Shalom *et al.* 2001; Kulikov and Kulikova 2014; Kulikova and Kulikov 2014). The CDEKF assumes that the underlying dynamic functions take on the form of a set of stochastic differential equation (SDE) functions as in Equation 1 and the measurements are available at discrete time intervals, as in Equation 4, but without the dependency on $\boldsymbol{S}_i(t)$, the latent regime indicator. In other words, the model handled by the CDEKF may be viewed as a single-regime special case of the general model shown in Equations 1–4.

For continuous processes in the form of Equation 1, let $\hat{\boldsymbol{\eta}}_i(t) = E(\boldsymbol{\eta}_i(t)|\mathbf{Y}_i(t_{i,j-1}))$ and $\mathbf{P}_i(t) = Cov[\boldsymbol{\eta}_i(t)|\mathbf{Y}_i(t_{i,j-1})]$ denote the mean and covariance matrix of the latent variables, respectively, at time t in the interval $[t_{i,j-1}, t_{i,j}]$. In the CDEKF framework, the prediction step of KF is replaced by solving a set of ordinary differential equations (ODE) at time $t_{i,j}$, given the initial conditions at time $t_{i,j-1}$: $\hat{\boldsymbol{\eta}}_i(t_{i,j-1}) = \hat{\boldsymbol{\eta}}_i(t_{i,j-1}|t_{i,j-1})$ and $\mathbf{P}_i(t_{i,j-1}) = \mathbf{P}_i(t_{i,j-1}|t_{i,j-1})$. This set of ODEs is obtained by only retaining the first term, $\boldsymbol{f}_{S_i(t)}\left(\hat{\boldsymbol{\eta}}_i(t), t, \boldsymbol{x}_i(t)\right)$, in the Taylor series expansion of $\boldsymbol{f}_{S_i(t)}\left(\boldsymbol{\eta}_i(t), t, \boldsymbol{x}_i(t)\right)$ around the expectation $\hat{\boldsymbol{\eta}}_i(t)$, and is shown below:

$$\frac{d\hat{\boldsymbol{\eta}}_i(t)}{dt} = \boldsymbol{f}_{S_i(t)}\left(\hat{\boldsymbol{\eta}}_i(t), t, \boldsymbol{x}_i(t)\right), \tag{7}$$

$$\frac{d\mathbf{P}_i(t)}{dt} = \frac{\partial \boldsymbol{f}_{S_i(t)}\left(\hat{\boldsymbol{\eta}}_i(t), t, \boldsymbol{x}_i(t)\right)}{\partial \hat{\boldsymbol{\eta}}_i(t)}\mathbf{P}(t) + \mathbf{P}(t)\left(\frac{\partial \boldsymbol{f}_{S_i(t)}\left(\hat{\boldsymbol{\eta}}_i(t), t, \boldsymbol{x}_i(t)\right)}{\partial \hat{\boldsymbol{\eta}}_i(t)}\right)^{\top} + \mathbf{Q}_{S_i(t)}. \tag{8}$$

where $\frac{\partial \boldsymbol{f}_{S_i(t)}(\hat{\boldsymbol{\eta}}_i(t), t, \boldsymbol{x}_i(t))}{\partial \hat{\boldsymbol{\eta}}_i(t)}$ is the Jacobian matrix of $\boldsymbol{f}_{S_i(t)}\left(\hat{\boldsymbol{\eta}}_i(t), t, \boldsymbol{x}_i(t)\right)$ with respect to $\hat{\boldsymbol{\eta}}_i(t)$ at time t. Kulikov and Kulikova (2014, Kulikova and Kulikov 2014) suggested solving for equations 7 and 8 using adaptive ODE solvers. We adopt one approximate numerical solution — the fourth-order Runge-Kutta (Press, Teukolsky, Vetterling, and Flannery 2002) method — to solve Equations 7 and 8. In cases where the hypothesized continuous-time dynamic functions are linear, explicit analytic solutions exist and there is no need to use numerical solvers. However, in our simulation work, estimating known special cases of linear stochastic differential equation models using numerical solvers yielded comparable estimates and computational time to estimating the same models using their known solutions. Thus, for generality, we utilize numerical solvers in solving both linear and nonlinear differential equations in **dynr**.

As in the case involving nonlinear discrete-time dynamic models, parameter estimates obtained from optimizing the log-likelihood function constructed from by-products of CDEKF are also approximate ML estimates; however, the approximations now stem both from the truncation errors from the first-order Taylor series expansion in the CDEKF, as well as the numerical solver that is used to solve Equations 7 and 8.

In cases involving regime-switching ordinary or stochastic differential equations, the algorithms for estimating regime-switching continuous-time models are essentially estimation procedures that combine the CDEKF and part of the Kim filter designed to handle estimation of the regime-switching portion of the model. The resultant procedure, referred to herein as *continuous-discrete extended Kim filter*, is summarized in Chow *et al.* (Under review).

		Discrete-time	Continuous-time
Single-regime	Linear	Linear state-space model KF **dynr**, **OpenMx**, **pomp**, **KFAS**, **dlm**, **dse**, **MKFM6**, **SsfPack**, MATLAB	Linear SDE/ODE CDEKF **dynr**, **pomp**, **OpenMx**, **ctsem**, MATLAB
	Nonlinear	Nonlinear state-space model EKF **dynr**, **pomp**, **SsfPack**, MATLAB	Nonlinear SDE/ODE CDEKF **dynr**, **pomp**, MATLAB
Multiple-regime	Linear	RS state-space model Kim filter **dynr**, GAUSS code, MATLAB	RS SDE/ODE CD Kim filter **dynr** only
	Nonlinear	RS nonlinear state-space model Extended Kim filter **dynr** only	RS nonlinear SDE/ODE CD extended Kim filter **dynr** only

Table 1: Models, algorithms, and software for the framework of regime-switching (non)linear state space models in discrete- and continuous-time. SDE = Stochastic Differential Equation, ODE = Ordinary Differential Equation, CD = Continuous-Discrete, RS = Regime-Switching, KF = Kalman filter (Kalman 1960), EKF = Extended Kalman filter (Anderson and Moore 1979; Bar-Shalom *et al.* 2001), Kim filter = KF + Hamilton filter + Collapsing procedure (Kim and Nelson 1999). Extended Kim filter was proposed by Chow and Zhang (2013); the CD extended Kim filter is proposed by Chow *et al.* (Under review).

4. Steps for preparing and "cooking" a model

The general procedure for using the **dynr** package can be summarized in five steps as below. The theme around the naming convention exploits the pronunciation of the package name: **dynr** is pronounced the same as "dinner." Therefore, the names of functions and methods are specifically designed to relate to things done surrounding dinner, such as gathering ingredients (e.g., the data), preparing recipes, cooking, which involves combining ingredients according to a "modeling" recipe and applies heat, and serving the finished product.

Implementation of a full "dinner" occurs as follows. First, data are gathered and identified with the `dynr.data()` function. Second, *recipes* are prepared. To each part of a model there is a corresponding `prep.*()` recipe function. Each of these functions creates an object of class `dynrRecipe`. Each `prep.*()` function creates an object of class `dynrThing` which is in turn a subclass of `dynrRecipe`. These recipe functions include:

1. The `prep.measurement()` function defines the measurement part of the model, that is, how latent variables and exogenous covariates map onto the observed variables.

2. The `prep.matrixDynamics()` and `prep.formulaDynamics()` functions define the dynamics of the model with either a strictly linear, matrix interface or with a possibly nonlinear formula interface, respectively.

3. The `prep.initial()` function defines the initial conditions of the model. The initial conditions are used by the iterative and recursive algorithms as the starting point for latent variable estimates. As such, the `prep.initial()` function describes the initial mean vector and covariance matrix of the latent variables, assumed to be multivariate normally distributed at the initial and all remaining time points.

4. The `prep.noise()` function defines the covariance structure for both the measurement (or observation) noise and the dynamic (or latent) noise.

5. The `prep.regimes()` function provides the regime switching structure of the model. Single-regime models do not require a `dynrRegimes` object.

Once the data and recipes are prepared, the third step mixes the data and recipes together into a model object of class `dynrModel` with the `dynr.model()` function. Fourth, the model is cooked with `dynr.cook()`. In **dynr**, cooking refers to estimating the free parameters and standard errors of a model. Fifth and finally, results are served in summary tables (with a `summary()` method), plots of trajectories and equations (with `plot()`, `dynr.ggplot()` (or its alias `autoplot()`), and `plotFormula()`), as well as LaTeXequations (with `printex()`).

We will demonstrate the interface of **dynr** using two examples of dynamic models that can be implemented using the package: (1) a linear state-space example with regime-switching properties based on Yang and Chow (2010); and (2) a regime-switching extension of the predator-prey model (Lotka 1925; Volterra 1926).

5. Example 1: Regime-switching linear state-space model

Facial electromyography (EMG) has been used in the behavioral sciences as one possible indicator of human emotions (e.g., Schwartz 1975; Cacioppo and Petty 1981; Cacioppo, Petty, Losch, and Kim 1986; Dimberg, Thunberg, and Elmehed 2000). When human subjects are exposed to emotion induction procedures, researchers have detected changes in individuals' facial EMG recordings even when the corresponding changes in facial expression are too subtle to be detected by human raters (Schwartz 1975; Dimberg 1990; Cacioppo and Petty 1981).

A time series of EMG data contains bursts of electrical activity that are typically magnified when an individual is under emotion induction. However, because data segments with bursts are very short and are interspersed with long periods of deactivation (i.e., segments without bursts), the distribution of EMG data deviates substantially from normality (see sample facial EMG data from one participant plotted in Figure 1(A)). To represent the patterns of EMG data, Yang and Chow (2010) proposed using a regime-switching linear state-space model in which the individual may transition between regimes with and without facial EMG activation. In this way, the data could be reasonably assumed to be normally distributed conditional on the regime in which an individual resides at a particular moment; heterogeneities in the dynamic patterns and variance of EMG data are also accounted for through the incorporation of these latent regimes. Model fitting was previously performed at the individual level. Data from the participant shown in Figure 1(A) are made available as part of the demonstrative examples in **dynr**. A complete modeling script for this example is available as one of the demo examples in **dynr** and can be run using `demo(RSLinearDiscreteYang)`. Here we present selected segments of code to showcase how a linear state-space model with regime-switching

can be specified in **dynr**. The model of interest is the final model selected for this participant by Yang and Chow (2010):

$$y_i(t_{i,j}) = \mu_{yS_i(t_{i,j})} + \beta_{S_i(t_{i,j})}\text{Self-report}(t_{i,j}) + \eta_i(t_{i,j}), \tag{9}$$
$$\eta_i(t_{i,j+1}) = \phi_{S_i(t_{i,j})}\eta_i(t_{i,j}) + \zeta_i(t_{i,j+1}), \tag{10}$$

in which we allowed the intercept, $\mu_{yS_i(t_{i,j})}$, the regression slope, $\beta_{S_i(t_{i,j})}$, and the autoregression coefficient, $\phi_{S_i(t_{i,j})}$, to be regime-dependent. By allowing $\phi_{S_i(t_{i,j})}$ to be regime-specific, we indirectly allowed the total variance of the latent component, $\eta_i(t_{i,j+1})$, to be heterogeneous across the deactivation and activation stages, in spite of requiring the dynamic noise variance, $E(\zeta_i(t)^2)$, to be constant across regimes.

The first step in **dynr** modeling is to structure the data. This is done with the `dynr.data()` function.

```
R> require("dynr")
R> data("EMG")
R> EMGdata <- dynr.data(EMG, id = 'id', time = 'time',
+  observed = 'iEMG', covariates = 'SelfReport')
```

The first argument of this function is either a `ts` class object of single-subject time series or a `data.frame` structured in a long (relational) format (i.e., with different measurement occasions from the same subject appearing as different rows in the data frame). Missing values in the observed variables should be indicated by `NA`. When a `ts` class object is passed to `dynr.data()`, no other inputs are needed. Otherwise, the `id` argument needs the name of the ID variable as input, and allows multiple people to be estimated in a single model by distinguishing different individuals with the ID variable. That is, it indicates which rows should be modeled together as a time series. Thus, multi-subject modeling is as easy as single-subject modeling; only the data differ. The `time` argument needs the name of the TIME variable that indicates subject-specific measurement occasions. If a discrete-time model is desired, the TIME variable should contain subject-specific sequences of (subsets of) consecutively equally spaced numbers (e.g, 1, 2, 3, $\cdots$). In other words, the program assumes that the input `data.frame` is equally spaced with potential missingness. If the measurment occasions for a subject are a subset of an arithmetic sequence but are not consecutive, `NA`s will be inserted automatically to create an equally spaced data set before estimation. If a continuous-time model is being specified, the TIME variable can contain subject-specific increasing sequences of irregularly spaced real numbers. That is, the data may be input at their original, irregularly spaced intervals without the need to insert missingness. In this particular example, a discrete time model is used.

The `observed` and `covariates` arguments are used to indicate the names of the observed variables and covariates in the data. Covariates are defined as fixed predictors that are hypothesized to affect the modeling functions in one or more ways, but are otherwise not of interest (i.e., not modeled as dependent variables) to the user. Missing values in covariates are not allowed. That is, missing values in the covariates, if there are any, should be imputed first. The `dynr.data()` function lets users include data sets with many variables, but only use a few. The output of the function combines with the model recipe information later to map the model onto the data.

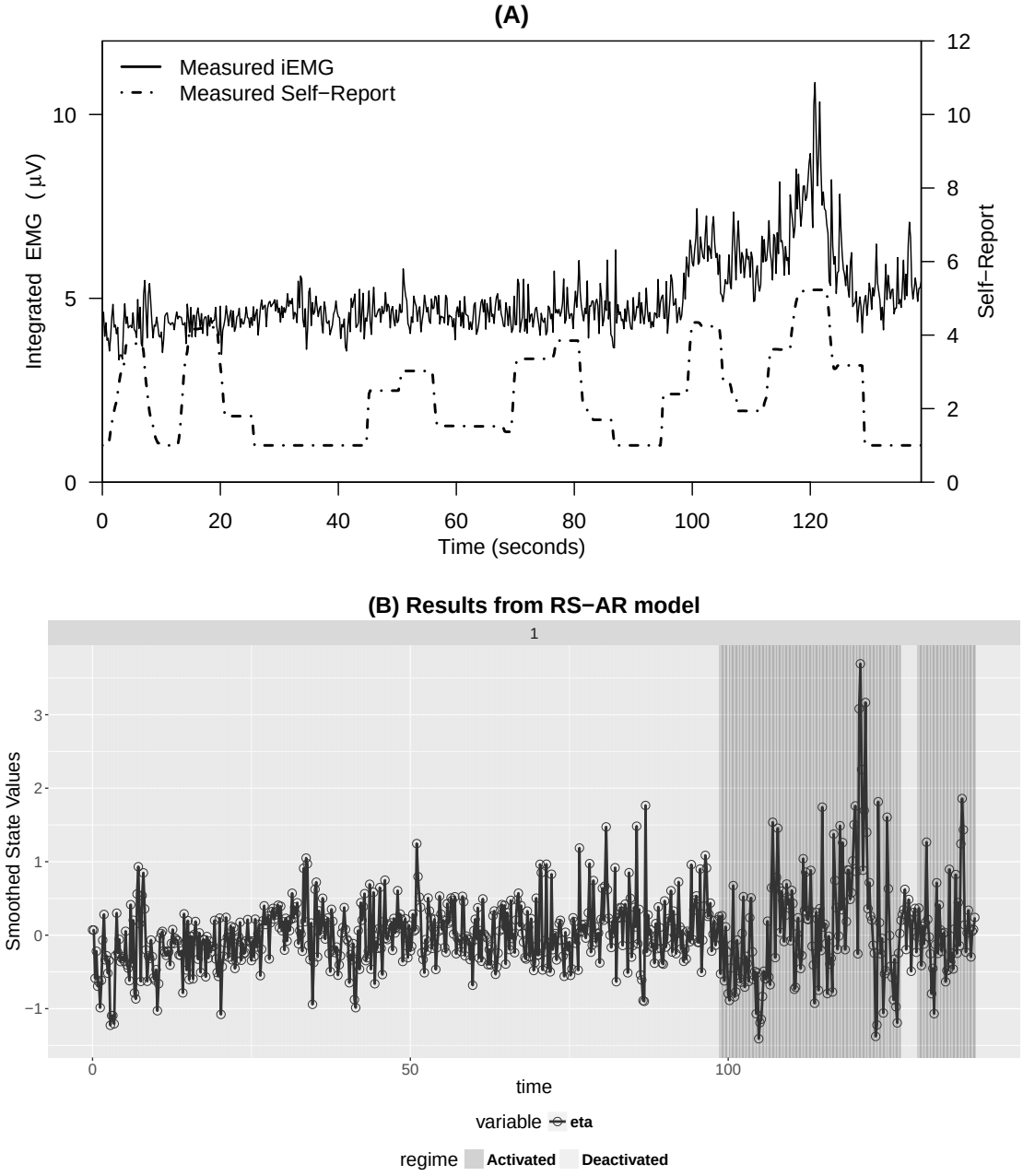

Figure 1: (A) A plot of integrated electromyography (iEMG) and self–report affect ratings for one participant with a time interval of 0.2 seconds between two adjacent observations. Self–report = self–report affect ratings; iEMG = integrated EMG signals. (B) An automatic plot of the smoothed state estimates for the regime-switching linear state-space model.

The next step in **dynr** modeling is to build the recipes for the various parts of a model. The recipes are created with `prep.*()` functions. For instance, the noise recipe is created with `prep.noise()`. The code below creates the noise recipe by calling the `prep.noise()` function. The noise recipe is stored in the `recNoise` object, an abbreviation for "recipe noise". The latent noise covariance matrix is a 1×1 matrix with a free parameter called `dynNoise`, short for "dynamic noise." The observed noise covariance matrix is also a 1×1 matrix, but has the measurement noise variance fixed to zero. These covariance matrices need to be positive definite. The zero's in the diagonal of a covariance matrix are internally replaced by a small positive number automatically before estimation. To ensure the matrix stays positive definite in estimation, we will apply a set of transformations to the matrix in each iteration of the optimization, so the starting or fixed values of these matrices are automatically adjusted for this purpose.

```
R> recNoise <- prep.noise(
+  values.latent = matrix(1, 1, 1),
+  params.latent = matrix('dynNoise', 1, 1),
+  values.observed = matrix(0, 1, 1),
+  params.observed = matrix('fixed', 1, 1))
```

A general feature of the `prep.*()` functions is to handle multiple regimes by using lists as the objects given to the recipe functions. The primary inputs of many of the recipe functions are the `values.*` and `params.*` arguments. The `values.*` arguments give the starting values and fixed values of parts of the recipe. The `params.*` arguments give the free parameter names corresponding to the values or indicate a fixed value by using the reserved name "fixed". The `values.*` and `params.*` arguments can be a single matrix, or a list of matrices for multiple-regime models with each element of a list corresponding to a regime.[3] If only one matrix is specified for a regime-switching dynamic model, the process noise covariance structure stays the same across regimes.

The next block of code creates the measurement recipe (i.e., the functions shown in Equation 4) by calling the `prep.measurement()` function. The measurement recipe is stored in the `recMeas` object, short for "recipe measurement". There are two important differences of interest between the code used for this function and that used for the noise recipe. First, instead of referring to `values.latent` and `values.observed` for the latent and observed noise, this function refers to `values.load`, `values.int`, and `values.exo` for the factor loadings, measurement intercepts, and exogenous covariate effects, respectively. Second, the measurement part of the model is regime-switching so the objects given to the `values.*` and `params.*` arguments are lists of matrices instead of matrices. The first element of each list corresponds to the first regime. The second element of each list is for the second regime.

```
R> recMeas <- prep.measurement(
+  values.load = rep(list(matrix(1, 1, 1)), 2),
+  values.int = list(matrix(4, 1, 1), matrix(3, 1, 1)),
+  params.int = list(matrix('mu_1', 1, 1), matrix('mu_2', 1, 1)),
+  values.exo = list(matrix(0, 1, 1), matrix(1, 1, 1)),
+  params.exo = list(matrix('fixed', 1, 1), matrix('beta_2', 1, 1)),
```

[3] Lists of length one are also acceptable, but they are more work to write. Internally, when single matrices are given to recipe functions they are turned into lists of length one.

```
+  obs.names = c('iEMG'),
+  state.names = c('eta'),
+  exo.names = c("SelfReport"))
```

The above code block creates the regime-switching measurement model where the factor loadings matrices for both regimes are 1×1 matrices fixed at one. Thus, the factor loadings are non-regime-switching. The two intercepts, corresponding to the μ_{yS_t} in Equation 9, are given starting values of four and three with the free parameter names `mu_1` and `mu_2`. The two covariate effects are given a similar pattern of starting values and correspond to the β_{S_t} of Equation 9, but the covariate effect in the Deactivated Regime is fixed to zero. This constraint implies that the self-reported emotions of the participant have no influence on their EMG data unless the participant is emotionally activated (i.e., in the Activated Regime). Lastly, the `*.names` arguments, including `obs.names`, `state.names`, and `exo.names`, give the names of the observed variables, latent states, and exogenous covariates, respectively.

The `prep.regimes()` function specifies the structure of the regime switching functions shown in Equation 6. Note that based on Equation 6, a total of $n_d + 1$ parameters, including an intercept, c_{lm}, and n_d regression slopes in $\mathbf{d}_{lm}$, have to be defined for each of the functions governing the transition from the lth regime ($l = 1, \ldots, M$) to the mth regime ($m = 1, \ldots, M$). In total, there are $M \times M$ of such transition functions, corresponding to entries in an $M \times M$ transition probability matrix. The function `prep.regimes()` requires the user to provide the starting values (through the `values` argument) and names (through the `params` argument) for these $M \times (n_d + 1)$ parameters as a matrix whose number of rows equals to the number of regimes (i.e., M) and number of columns equals to the product of the number of regimes and the total number of parameters (i.e., $(n_d + 1)M$) as:

$$\left[\begin{array}{cc:cc:c:cc} c_{11} & \mathbf{d}_{11}^\top & c_{12} & \mathbf{d}_{12}^\top & \cdots & c_{1M} & \mathbf{d}_{1M}^\top \\ \hdashline c_{21} & \mathbf{d}_{21}^\top & c_{22} & \mathbf{d}_{22}^\top & \cdots & c_{2M} & \mathbf{d}_{2M}^\top \\ \hdashline \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hdashline c_{M1} & \mathbf{d}_{M1}^\top & c_{M2} & \mathbf{d}_{M2}^\top & \cdots & c_{MM} & \mathbf{d}_{MM}^\top \end{array}\right]. \tag{11}$$

In this example, we do not have any covariates in the regime-switching (RS) functions. Thus, all the cells corresponding to the entries in $\mathbf{d}_{lm}$ drop out. The problem then reduces to the specification of a 2×2 transition log-odds (LO) matrix. Here, we are interested in specifying a RS model in which conditional on any of the two previous regimes, there is a higher probability of staying in the current regime than transitioning to a different regime. To accomplish this, we first note that we set the LO entries in second column of the 2×2 transition LO matrix (corresponding to the Activated Regime) to zero for identification purposes. Thus, the Activated Regime serves in this case as the reference regime. The first column of the transition LO matrix, which consists of freely estimated LO parameters named `c11` and `c21`, is populated with the starting values of: (1) `c11` $= 0.7$, corresponding to $exp(0.7) = 2.01$ times greater LO of staying within the Deactivated Regime as transitioning from the Deactivated to the Activated Regime, the reference regime; and (2) `c21` $= -1$, corresponding to $exp(-1) = 0.37$ times lower LO of transitioning from the Activated Regime to the Deactivated Regime relative to the LO of staying Activated.

```
R> recReg <- prep.regimes(
+  values = matrix(c(.7, -1, 0, 0), 2, 2),
+  params = matrix(c('c11', 'c21', 'fixed', 'fixed'), 2, 2))
```

In essence, the above code creates the following transition probability matrix:

$$\begin{array}{c} \\ Deactivated_{t_{i,j}} \\ Activated_{t_{i,j}} \end{array} \begin{array}{c} \begin{array}{cc} Deactivated_{t_{i,j+1}} & Activated_{t_{i,j+1}} \end{array} \\ \begin{pmatrix} \frac{exp(c_{11})}{exp(c_{11})+exp(0)} & \frac{exp(0)}{exp(c_{11})+exp(0)} \\ \frac{exp(c_{21})}{exp(c_{21})+exp(0)} & \frac{exp(0)}{exp(c_{21})+exp(0)} \end{pmatrix} \end{array} \tag{12}$$

with starting values of

$$\begin{array}{c} \\ Deactivated_{t_{i,j}} \\ Activated_{t_{i,j}} \end{array} \begin{array}{c} \begin{array}{cc} Deactivated_{t_{i,j+1}} & Activated_{t_{i,j+1}} \end{array} \\ \begin{pmatrix} .668 & .332 \\ .269 & .731 \end{pmatrix}. \end{array} \tag{13}$$

In cases where covariates are involved in the multinomial logistic regression, a `covariates` argument allows us to provide the names of the covariates according to the order of the elements in $\mathbf{d_{lm}}$. The second example shows equations and code that illustrate how covariates can be incorporated into the multinomial logistic regression (e.g., Equations 23 and 24 and neighboring blocks of code).

In many situations it is useful to specify the structure of the transition LO matrix in deviation form - that is, to express the LO intercepts in all but the reference regime as deviations from the LO intercept in the reference regime. This creates a comparison class with all other transition intercepts evaluated as compared to that class. Note that the deviation reference regime differs from that described previously. The former description had only a reference *column*, whereas the deviation reference regime adds to this a reference *row*. In the deviation case it is expedient to reformulate the intercept as the sum of a baseline and a deviation:

$$c_{lm} = c_m + c_{\Delta,lm} \tag{14}$$

where c_m denotes the logit intercept for the probability of switching into latent class m from the reference *row* class, $c_{\Delta,lm}$ denotes the deviation in LO of switching into latent class m at time $t_{i,j}$ from latent class l (i.e., from $S_i(t_{i,j-1}) = l$ to $S_i(t_{i,j}) = m$), as compared to switching from the reference *row* class. In this case, the multinomial logistic regression equation in Equation 6 now appears as:

$$\Pr\left(S_i(t_{i,j}) = m | S_i(t_{i,j-1}) = l, \boldsymbol{x}_i(t_{i,j})\right) \quad \stackrel{\Delta}{=} \pi_{lm,it} = \frac{\exp(c_m + c_{\Delta,lm} + \mathbf{d}_{lm}^T \boldsymbol{x}_i(t_{i,j})}{\sum_{k=1}^{M} \exp(c_k + c_{\Delta,lk} + \mathbf{d}_{lk}^T \boldsymbol{x}_i(t_{i,j}))}, \tag{15}$$

and all parameters in Equation 15 can be summarized into

$$\left[\begin{array}{cc:cc:c:cc} c_{\Delta,11} & \mathbf{d}_{11}^T & c_{\Delta,12} & \mathbf{d}_{12}^T & \cdots & c_{\Delta,1M} & \mathbf{d}_{1M}^T \\ \hdashline c_{\Delta,21} & \mathbf{d}_{21}^T & c_{\Delta,22} & \mathbf{d}_{22}^T & \cdots & c_{\Delta,2M} & \mathbf{d}_{2M}^T \\ \hdashline \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hdashline c_1 & \mathbf{d}_{M1}^T & c_2 & \mathbf{d}_{M2}^T & \cdots & c_M & \mathbf{d}_{MM}^T \end{array}\right], \tag{16}$$

if we use regime M for the reference row and hence have $c_{\Delta,Ml} = 0$ for $l = 1, \ldots, M$. This allows the same parameter matrix structure for both the deviation form (Equation 16) and non-deviation form (Equation 11) of the regime switching probabilities by dropping the $c_{\Delta,Ml}$ terms in the reference row and replacing them with the c_m logit intercepts. For convenience, both the deviation and the non-deviation form (Equations 15 and 6) are available in **dynr**.

That is, users are allowed to estimate either parameter matrix 11 or 16. For identification purposes, we can again choose regime M as the reference column and impose the constraints that $c_M = c_{\Delta,lM} = 0$ and $\mathbf{d}_{lM} = \mathbf{0}$ for $l = 1, \ldots, M$ to ensure that $\sum_{m=1}^{M} \pi_{lm} = 1$. Likewise, above we have shown an example that uses regime M for the reference row and column, but these are independent choices that can be made by the user. The following code specifies a deviation form for this example.

```
R> recReg2 <- prep.regimes(
+  values = matrix(c(.8, -1, 0, 0), 2, 2),
+  params = matrix(c('c_Delta11', 'c1', 'fixed', 'fixed'), 2, 2),
+  deviation = TRUE, refRow = 2)
```

By default the reference row is set to the automatically detected reference column, but the code makes this choice explicit. Importantly, this code creates the same starting values as seen in Equation 13 but parameterized in the form of Equation 16. The deviation form can be extremely useful for testing hypotheses about the relationships between LO intercepts and for making constraints across regimes.

The dynamic functions, $\boldsymbol{f}_{S_i(t)}()$ in Equations 1 and 2, can be specified using one of two possible functions in **dynr**: `prep.formulaDynamics()` and `prep.matrixDynamics()`. The dynamic model in this particular example consists only of linear functions, although the parameters that appear in these linear functions are regime-dependent. In this special case, the dynamic model in Equation 2 reduces to:

$$\boldsymbol{\eta_i}(t_{i,j+1}) = \boldsymbol{\alpha}_{S_i(t_{i,j})} + \boldsymbol{F}_{S_i(t_{i,j})}\boldsymbol{\eta_i}(t_{i,j}) + \boldsymbol{B}_{S_i(t_{i,j})}\boldsymbol{x}_i(t_{i,j}) + \boldsymbol{w}_i(t_{i,j+1}), \tag{17}$$

where the general, possibly nonlinear function $\boldsymbol{f}_{S_i(t)}()$ is replaced with a linear function consisting of (1) an intercept term $\boldsymbol{\alpha}_{S_i(t_{i,j})}$, (2) linear dynamics instantiated as an $r \times r$ matrix $\boldsymbol{F}_{S_i(t_{i,j})}$, (3) linear covariate regression effects $\boldsymbol{B}_{S_i(t_{i,j})}$, and the same additive noise term $\boldsymbol{w}_i(t_{i,j+1})$. As indicated by the subscript $S_i(t_{i,j})$, all of these can also be regime-dependent. Of course, the same structure is possible in continuous time as the linear analog of Equation 1.

$$d\boldsymbol{\eta_i}(t) = \left(\boldsymbol{\alpha}_{S_i(t)} + \boldsymbol{F}_{S_i(t)}\boldsymbol{\eta_i}(t) + \boldsymbol{B}_{S_i(t)}\boldsymbol{x}_i(t)\right) dt + d\boldsymbol{w}_i(t), \tag{18}$$

In this example, the dynamics as in Equation 10 are linear and discrete-time, so we can describe the dynamics in terms of Equation 17 as

$$\eta_i(t_{i,j+1}) = \underbrace{0}_{\boldsymbol{\alpha}_{S_i(t_{i,j})}} + \underbrace{\phi_{S_i(t_{i,j})}}_{\boldsymbol{F}_{S_i(t_{i,j})}}\eta_i(t_{i,j}) + \underbrace{0}_{\boldsymbol{B}_{S_i(t_{i,j})}}\boldsymbol{x}_i(t_{i,j}) + \underbrace{\zeta_i(t_{i,j+1})}_{\boldsymbol{w}_i(t_{i,j+1})}. \tag{19}$$

The `prep.matrixDynamics()` function allows the user to specify the structures of the intercept vector $\boldsymbol{\alpha}_{S_i(t_{i,j})}$, through `values.int` and `params.int`, the covariate regression matrix $\boldsymbol{B}_{S_i(t_{i,j})}$, through `values.exo` and `params.exo`, and the one-step-ahead transition matrix $\boldsymbol{F}_{S_i(t_{i,j})}$, through `values.dyn` and `params.dyn`, in the linear special case for those who prefer to work in such a matrix algebraic framework. We illustrate this function in the current example below. The `values.dyn` argument gives a list of matrices for the starting values of $\boldsymbol{F}_{S_i(t_{i,j})}$. The `params.dyn` argument names the free parameters. These are the ϕ_{S_t} in Equation 10. The `isContinuousTime` argument switches between continuous-time modeling

(when true) and discrete-time modeling (when false). Because this argument is false, the dynamics are in a discrete-time form that matches Equation 10. The arguments corresponding to the intercepts (`values.int` and `params.int`) and the covariate effects (`values.exo` and `params.exo`) are omitted to leave these matrices as zeros. Later examples will show how to specify continuous-time nonlinear dynamics.

```
R> recDyn <- prep.matrixDynamics(
+  values.dyn = list(matrix(.1, 1, 1), matrix(.5, 1, 1)),
+  params.dyn = list(matrix('phi_1', 1, 1), matrix('phi_2', 1, 1)),
+  isContinuousTime = FALSE)
```

After the recipes for all parts of the model are defined, the `dynr.model()` function creates the model and stores it in the `dynrModel` object. Each recipe (i.e., objects of class `dynrRecipe` created by `prep.*()`) and the data prepared by `dynr.data()` are given to this function. The function requires `dynamics`, `measurement`, `noise`, `initial`, and `data` as mandatory inputs for all models. When there are multiple regimes in the model, the `regimes` argument should be provided as shown below. When parameters are subject to transformation functions, a `transform` argument can be added, which will be discussed in the second example. The `dynr.model()` function takes the recipes and the data and combines information from both. In doing so, this function uses the information from each recipe to write the text for a C function. Optionally, the C functions can be written to a file named by the `outfile` argument (i.e., "RSLinearDsicrete.c" in this specific example) so that the user can inspect the automatically generated C code. Ideally of course, there is no need to ever examine this file; however, it is sometimes useful for debugging purposes and may be helpful for specifying models that extend those supported by the R interface functions. More frequently, inspecting the `dynrModel` object and "serving it" will provide the needed information.

```
R> rsmod <- dynr.model(
+  dynamics = recDyn,
+  measurement = recMeas,
+  noise = recNoise,
+  initial = recIni,
+  regimes = recReg,
+  data = EMGdata,
+  outfile = "RSLinearDiscrete.c")

R> yum <- dynr.cook(rsmod)
```

In the last line above, the model is "cooked" with the `dynr.cook()` function to estimate the free parameters and their standard errors. When cooking, the C code that was written by `dynr.model()` is compiled and dynamically linked to the rest of the compiled **dynr** code. Then the C is executed to optimize the free parameters while calling the dynamically linked C functions that were created from the user-specified recipes. There are two points worth emphasizing in this regard. First, the user never has to write C functions. Second, the user benefits from the C functions because of their speed. In this way, **dynr** provides an R interface for dynamical systems modeling while maintaining much of the speed associated with C.

The final step associated with **dynr** modeling is serving results (a `dynrCook` object) after the model has been cooked. To this end, several standard, popular S3 methods are defined for

the `dynrCook` class, including `coef()`, `confint()`, `deviance()`, `logLik()` (and thus implicitly `AIC()` and `BIC()`), `names()`, `nobs()`, `summary()`, and `vcov()`. These methods perform the same tasks as their counterparts for regression models (i.e., `lm` class objects). Besides, **dynr** also provides a few other model-serving functions. Here we illustrate in turn: `summary()`, `plot()`, `dynr.ggplot()` (or `autoplot()`), `plotFormula()`, and `printex()`. The `summary()` method provides a table of free parameter names, estimates, standard errors, t-values, and Wald-type confidence intervals.

```
R> summary(yum)

Coefficients:
         Estimate Std. Error t value ci.lower ci.upper Pr(>|t|)
phi_1     0.26608    0.04953   5.372  0.16900  0.36315 5.33e-08 ***
phi_2     0.47395    0.04425  10.711  0.38722  0.56068  < 2e-16 ***
beta_2    0.46449    0.04394  10.571  0.37837  0.55061  < 2e-16 ***
mu_1      4.55354    0.02782 163.658  4.49901  4.60807  < 2e-16 ***
mu_2      4.74770    0.14250  33.318  4.46842  5.02699  < 2e-16 ***
dynNoise  0.20896    0.01129  18.504  0.18683  0.23110  < 2e-16 ***
c11       5.50199    0.70939   7.756  4.11160  6.89237  < 2e-16 ***
c21      -5.16170    1.00424  -5.140 -7.12998 -3.19342 1.79e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

-2 log-likelihood value at convergence = 1002.52
AIC = 1018.52
BIC = 1054.87
```

These parameter estimates, standard errors, and likelihood values closely mirror those reported in Yang and Chow (2010, p. 755-756). In the Deactivated Regime, the autoregressive parameter (`phi_1`) and the intercept (`mu_1`) are lower than in the Activated Regime. So, neighboring EMG measurements are more closely related in the Activated Regime and the overall level is slightly higher. This matches very well with the idea that the Activated Regime consists of bursts of facial muscular activities and an elevated emotional state. Similarly, the effect of the self-reported emotional level is positive in the Activated Regime and fixed to zero in the Deactivated Regime. In the nested model that freely estimated this covariate effect in the Deactivated Regime, it was estimated at -0.00258 with a t-value of -0.097 and thus was subsequently fixed at zero. So, in the Deactivated Regime there is no relationship between the self-reported emotional level and the facial muscular activity. Essentially, in the Activated Regime the facial EMG and the self-reported emotions become coupled, but in the Deactivated Regime they are unrelated. The dynamic noise parameter gives a sense of the size of the intrinsic unmeasured disturbances that act on the system. These forces perturb the system with a typical magnitude (i.e., standard deviation) of a little less than half a point on the EMG scale seen in Figure 1(A). Lastly, the log-odds parameters (`c11` and `c21`) can be turned into the transition probability matrix yielding

$$\begin{array}{c} \\ Deactivated_{t_{i,j}} \\ Activated_{t_{i,j}} \end{array} \begin{array}{c} \begin{array}{cc} Deactivated_{t_{i,j+1}} & Activated_{t_{i,j+1}} \end{array} \\ \begin{pmatrix} .9959 & .0041 \\ .0057 & .9943 \end{pmatrix} \end{array} \tag{20}$$

which implies that both the Deactivated and the Activated Regimes are strongly persistent with high self-transistion probabilities. Next we consider some of the visualization options for serving a model.

The default `plot()` method is used to visualize the time series in a collection of plots: (1) a plot of time series created by `dynr.ggplot()` (or `autoplot()`), (2) a histogram of predicted regimes, and (3) a plot of equations created by `plotFormula()`.

```
R> plot(yum, dynrModel = rsmod, style = 1, textsize = 5)
```

The `dynr.ggplot()` (or `autoplot()`) method creates a plot of the smoothed state estimates overlaying the predicted regimes. It needs the result object and model object as inputs, and allows for plotting (1) user-selected smoothed state variables by default and (2) user-selected observed-versus-predicted values by setting a `style` to 2. An illustrative plot is created from the code below and shown in Figure 1(B).

```
R> dynr.ggplot(yum, dynrModel = rsmod, style = 1,
+  names.regime = c("Deactivated", "Activated"),
+  title = "(B) Results from RS-AR model", numSubjDemo = 1,
+  shape.values = c(1),
+  text = element_text(size = 16),
+  is.bw = TRUE)
```

This shows that for the first 99 seconds the participant is in the Deactivated Regime, with their EMG data varying according to the lower autocorrealtion model and having no relation to the variation in the self-reported emotional data in Figure 1(A). Then the participant switches to the Activated Regime and their data become more strongly autocorrelated and coupled to the self-report data. There follows a brief period in the Deactivated Regime around time=130 seconds with a subsequent return to the Activated Regime for the remainder of the observation.

For all users, the `plotFormula()` method can be used to display equations on R plots. Equations can be viewed in several ways after the model is specified: (1) with free parameter names and fixed values, as illustrated here in Figure 2(A), (2) with parameter starting values, or (3) after estimation with fitted parameter values as in Figure 2(B). Each of these desired characteristics can be embedded in the neatly typeset equations. The `ParameterAs` argument changes which of these characteristics is used in the equations. Here the user-supplied parameter names and estimated parameters are typeset in Figure 2 because `ParameterAs` was respectively given `rsmod$param.names`, namely, the parameter names stored in the `dynrModel` object, `rsmod`, and `coef(yum)`, namely, the estimated free parameter values stored in the `dynrCook` object, `yum`. Starting values for parameters are also possible values for this argument. The `plotFormula()` method does not require the user to install LaTeX facilities and compile LaTeX code in a separate step, and hence are convenient to use. To maximize the readability of the equations, it is only shown here using equations for the dynamic model and measurement model, which can be obtained by respectively setting the `printDyn` and `printMeas` arguments to true.

```
R> plotFormula(dynrModel = rsmod, ParameterAs = rsmod$param.names,
+  printDyn = TRUE, printMeas = TRUE) +
```

```
+  ggtitle("(A)") +
+  theme(plot.title = element_text(hjust = 0.5, vjust = 0.01, size = 16))

R> plotFormula(dynrModel = rsmod, ParameterAs = coef(yum),
+  printDyn = TRUE, printMeas = TRUE) +
+  ggtitle("(B)") +
+  theme(plot.title = element_text(hjust = 0.5, vjust = 0.01, size = 16))
```

We can see that the equations in Figure 2(A) are precisely those from Equations 9 and 10 which we used to define the model except that we have fixed β_1 to zero. If these equations did not match, it may indicate that we made a mistake in our model specification.

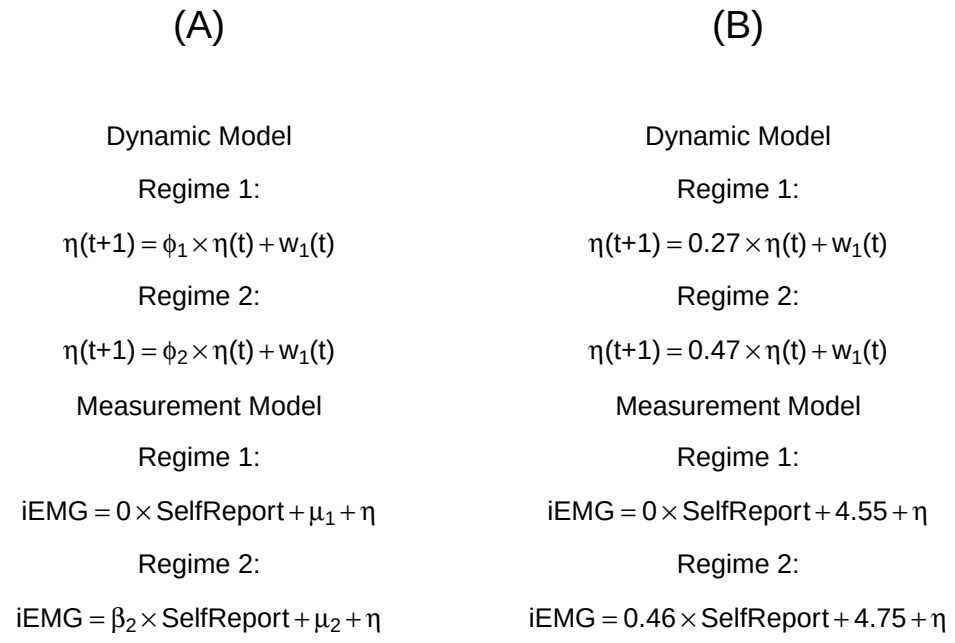

Figure 2: Automatic plots of model equations with (A) parameter names and (B) estimated parameters for the regime-switching linear state-space model.

Finally, for LaTeX users, the `printex()` method helps generate equations for the model in LaTeX form.

```
R> printex(rsmod,
+  ParameterAs = rsmod$param.names,
+  printInit = TRUE, printRS = TRUE,
+  outFile = "RSLinearDiscreteYang.tex")
```

The `ParameterAs` argument functions the same as that in the `plotFormula()` method. Here we have selected to use the names of the free parameters as evidenced by giving `rsmod$param.names` to the `ParameterAs` argument. In this case the initial conditions and

regime-switching functions are included in the equations, as indicated by the `printInit` and `printRS` arguments being set to true. The LaTeX code for the equations is written to the file specified, "RSLinearDiscrete.tex", which the user can then work with and modify as he/she wishes. Of course, this function is designed more as a convenience feature for users who are already using LaTeX as a writing tool and requires all the LaTeX-related facilities already in place on the user's computer. If so desired, the tex file can also be compiled within R and viewed as a pdf via the `texi2pdf()` function in the **tools** library:

```
R> tools::texi2pdf("RSLinearDiscreteYang.tex")
R> system(paste(getOption("pdfviewer"), "RSLinearDiscreteYang.pdf"))
```

This example has used real EMG data from a previous study (Yang and Chow 2010) to illustrate many parts of the user-interface for **dynr**. A complete working example can be found as part of the package demos using `demo(RSLinearDiscreteYang)`. Of particular note are the various "serving" functions which allow users to both verify their model and examine their results in presentation-ready formats. In the next example, we will use simulated data to further illustrate features of **dynr**, especially the nonlinear formula interface for dynamics.

6. Example 2: Nonlinear continuous-time models

As extensions of linear models, nonlinear dynamic models incorporate nonlinearities into the change processes. Such nonlinearities may take the form of interactions between components of a system, and have many useful applications across different scientific disciplines. In the study of human dynamics, for instance, many processes are characterized by changes that are dependent on interactions with other processes. Nonlinear ordinary differential equations have been used to model, among other phenomena, ovulatory regulation (Boker, Neale, and Klump 2014), circadian rhythms (Brown and Luithardt 1999), cerebral development (Thatcher 1998), substance use (Boker and Graham 1998), cognitive aging (Chow and Nesselroade 2004), parent-child interactions (Thomas and Martin 1976), couple dynamics (Chow *et al.* 2007; Gottman 2002); and sudden transitions in attitudes (van der Maas, Kolstein, and van der Pligt 2003).

6.1. Single-regime nonlinear continuous-time model

To facilitate the specification of more complex dynamic models, especially those that involve the use of specialized mathematical functions (e.g., trigonometric, power, logistic and exponential functions), or those for which the user would rather not specify in matrix form, **dynr** provides users with a formula interface that can accommodate nonlinear as well as linear dynamic functions. To illustrate the use of the formula interface in **dynr**, we use a benchmark nonlinear ordinary differential equation model, the predator-prey model (Lotka 1925; Volterra 1926; Hofbauer and Sigmund 1988). The predator-prey model is a classic model for representing the nonlinear dynamics of interacting populations or components of any system of interest. In this model, there are two populations, one of predators (e.g., foxes) and another of prey (e.g., rabbits). The food supply of the prey is assumed to be unbounded, but the food supply of the predators is the prey. As the predator population grows, they decrease the prey population. Consequently, as the prey population shrinks, the predator population must also decrease with its diminishing food supply. The most often cited behavior of the predator-prey

system while in a particular parameter range is ongoing oscillations in the predator and prey populations with a phase lag between them.

The utility of the predator-prey model extends far beyond the area of population dynamics. Direct applications or extensions of this predator-prey system include the epidemic models of the onset of social activities (EMOSA) used to study the spread of smoking, drinking, delinquency, and sexual behaviors among adolescents (Rodgers and Rowe 1993; Rodgers, Rowe, and Buster 1998), the cognitive aging model (Chow and Nesselroade 2004), and the model of couples' affect dynamics (Chow *et al.* 2007). In the EMOSA, smokers (predators) may interact with non-smokers (prey) to produce varying numbers of smokers and nonsmokers over time depending on the parameters of the system. Likewise, romantic couples may mutually drive their partners' affective states through ongoing interactions with each other, creating novel and testable hypotheses about human behavior.

Written as a differential equation, the predator-prey model is expressed as:

$$d(prey(t)) = (a\ prey(t) - b\ prey(t)\ predator(t))\, dt \tag{21}$$

$$d(predator(t)) = (-c\ predator(t) + d\ prey(t)\ predator(t))\, dt \tag{22}$$

where the parameters a, b, c, d are all constrained to be greater than or equal to 0. These equations make up the continuous-time dynamics for this system (i.e., the special case of Equation 1 for this model). Examining the prey equation (Equation 21), the prey population would increase exponentially without bound if there were zero predators. Similarly, examining the predator equation (Equaltion 22), if the prey population was zero, then the predator population would decrease exponentially to zero (i.e., go extinct).

Using the formula interface in **dynr**, which supports all native mathematical functions available in R, the predator-prey model can be specified as:

```
R> preyFormula <- prey ~ a * prey - b * prey * predator
R> predFormula <- predator ~ - c * predator + d * prey * predator
R> ppFormula <- list(preyFormula, predFormula)
R> ppDynamics <- prep.formulaDynamics(formula = ppFormula,
+  startval = c(a = 2.1, c = 0.8, b = 1.9, d = 1.1),
+  isContinuousTime = TRUE)
```

The first argument of the `prep.formulaDynamics()` function is `formula`. More specifically, this is a list of formulas. Each element in the list is a single, univariate, formula that defines a differential (if `isContinuousTime` = TRUE) or difference (if `isContinuousTime` = FALSE) equation. There should be one formula for every latent variable, in the order in which the latent variables are specified by using the `state.names` argument in `prep.measurement()`. The left-hand side of each formula is either the one-step-ahead projection of the latent variable (in the discrete-time case) or the differential of the latent variable (in the continuous-time case), namely, the left-hand-side of Equations 2 and 1, respectively. In both cases, users only need to specify the names of the latent variables that match the specification in `prep.measurement()` on the left-hand side of the formulas. The right-hand side of each formula gives a (linear or possibly nonlinear) function that may involve free or fixed parameters, numerical constants, exogenous covariates, and other arithmetic/mathematical functions that define the dynamics of the latent variables. The `startval` argument is a named vector giving the names of the free parameters and their starting values. Just as in the `prep.matrixDynamics()` function,

the `isContinuousTime` argument is a binary flag that defines the switch between continuous- and discrete-time modeling. The rest of **dynr** code for fitting the predator-prey model can be specified in similar ways to the code shown in Example 1 and is omitted here for space constraints. A fully functional demo script can be found as one of the demos in **dynr** using `demo(NonlinearODE)`.

With the formula interface, it is important to note that **dynr** uses the `D()` function from the **stats** package to automatically and symbolically differentiate the formulas provided. Hence, **dynr** uses the analytic Jacobian of the dynamics in its extended Kalman filter, greatly increasing its speed and accuracy. The `D()` function can handle the differentiation of functions involving parentheses, arithmetic operators (e.g., $+$, $-$, $*$, $/$, and $\hat{}$) and numerous mathematical functions (e.g., `exp`, `log`, `sin`, `cos`, `tan`, `sinh`, `cosh`, `sqrt`, `pnorm`, `dnorm`, `asin`, `acos`, `atan`, `gamma`, and so on). Thus, for a very large class of nonlinear functions, the user is spared from the need to supply the analytic Jacobian of the dynamic functions of interest to use the extended Kalman filter functionality in **dynr**. However, automatic differentiation will not work for all formulas. For instance, formulas involving the absolute value function cannot be symbolically differentiated. For formulas that cannot be differentiated automatically using the **stats** package, the user must provide the analytic first derivatives through the `jacobian` argument (check `demo(RSNonlinearDiscrete)` for an example).

6.2. Regime-switching extension

Just as with the `prep.matrixDynamics()`, the formula interface also allows for regime-switching functionality. Consider an extension of the classical predator-prey model. It is likely that the prey and predator interaction follow seasonal patterns. Hypothetically, we assume that in warmer seasons (i.e., "summer" environment), the interactions follow a classical predator-prey model, but in colder seasons (i.e., "winter" environment), the food source (e.g., grass) of the prey becomes limited and the predator species is able to find an additional food source due to the weather. So in the colder seasons the prey or predator population will not go to extreme values in absence of the other species. We thus consider the following regime-switching version of the predator-prey model to capture the potential seasonal changes in the interaction patterns. In the Summer regime, we have the predator-prey model as previously described, but in the Winter regime we now have a predator-prey model characterized by within-species competition and limiting growth/decay. In this competitive predator-prey model, the two populations do not grow/decline exponentially without bound in absence of the other, but rather, they grow logistically up to some finite carrying capacity. This logistic growth adds to the between-species interactions with the other population. This model can be specified as:

```
R> cPreyFormula <- prey ~ a * prey - e * prey ^ 2 - b * prey * predator
R> cPredFormula <- predator ~
+  f * predator - c * predator ^ 2 + d * prey * predator
R> cpFormula <- list(cPreyFormula, cPredFormula)
```

where the predator and prey equations are combined and supplied as a list.

To specify the regime-switching predator-prey model, we combine the classical predator-prey model and the predator-prey model with within-species competition into a list of lists. Then we provide this list to the usual `prep.formulaDynamics()` function as the `formula` argument.

```
R> rsFormula <- list(ppFormula, cpFormula)
R> dynm <- prep.formulaDynamics(formula = rsFormula,
+  startval = c(a = 2.1, c = 3, b = 1.2, d = 1.2, e = 1, f = 2),
+  isContinuousTime = TRUE)
```

We have simulated the data with true parameter values: $a = 2, b = 1, c = 4, d = 1, e = .25, f = 5$. The phase portraits of the classical predator-prey model (Summer regime) and the cometitive predator-prey model (i.e., Winter regime) are shown in Figure 3 created by the **phaseR** R package (Grayling 2014), where the two axes respectively represent the population size of the two species. In Figure 3(A), there is a reciprocal relation between the prey and predator population, whereas in Figure 3(B), there is an attractor or equilibrium state at $(1.5, 1.625)$, toward which the system tends to evolve.

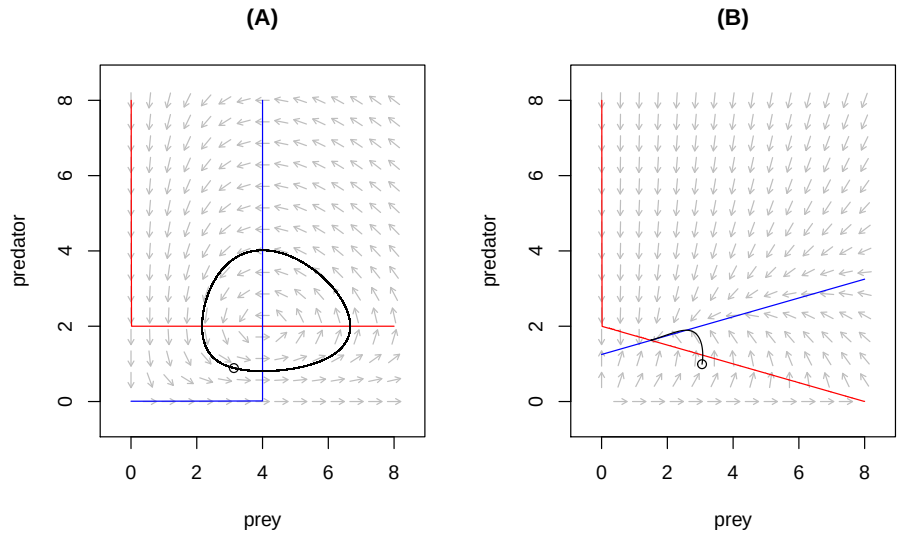

Figure 3: The phase portraits of (A) a classical predator-prey model and (B) a predator-prey model with within-species competition and limiting growth/decay.

Many dynamic models may only lead to permissible (e.g., finite) values in particular parameter ranges. As such, we often need to add constraints to model parameters in fitting dynamic models. One way of doing this in **dynr** is to apply unconstrained optimization while transforming the parameters onto their constrained scales during function evaluations. This can be accomplished in **dynr** through the function `prep.tfun()`. For example, based on the nature of the predator and prey dynamics, the a-f parameters should, by right, take on positive values. Thus, we may choose to optimize their log-transformed values and exponentiate the unconstrained parameter values during likelihood evaluations to ensure that the values of these parameter estimates are always positive. To achieve this, we supply a list of transformation formulas to the `formula.trans` argument in the prep.tfun() function as follows:

```
R> tformList <- list(a ~ exp(a), b ~ exp(b), c ~ exp(c),
+  d ~ exp(d), e ~ exp(e), f ~ exp(f))
```

```
R> tformInvList <- list(a ~ log(a), b ~ log(b), c ~ log(c),
+  d ~ log(d), e ~ log(e), f ~ log(f))
R> trans<-prep.tfun(
+  formula.trans = tformList,
+  formula.inv = tformInvList)
```

In cases involving the use of such constraint functions, the delta method is used to perform appropriate transformations to the covariance matrix of the parameter estimates at convergence to yield standard error estimates for the parameters on the constrained scales. If the starting values of certain parameters are indicated on a constrained scale, the `formula.inv` argument should give a list of inverse transformation formulas to transform the specified starting values to unconstrained scales for optimization.

In our hypothetical example, we have discussed how the weather condition may govern the regime switching processes. Specifically, we assume a covariate `cond` (with a value of 0 indicating the warmer weather and 1 indicating the colder weather) has an effect on the regime-switching transition probabilities. Then, we can specify the logistic regression model by

```
R> rmat <- matrix(
+  c(0, 0, -1, 1.5,
+  0, 0, -1, 1.5),
+  nrow = 2, ncol = 4, byrow = T)
R> pmat <- matrix(
+  c("fixed", "fixed", "int_1", "slp_1",
+  "fixed", "fixed", "int_2", "slp_2"),
+  nrow = 2, ncol = 4, byrow = T)
R> regimes <- prep.regimes(
+  values = rmat,
+  params = pmat,
+  covariates = "cond")
```

In essence, the above code creates the following matrix in the form of Equation 11

$$\left[\begin{array}{cc:cc} c_{11}=0 & d_{11}=0 & c_{12}=int_1=-1 & d_{12}=slp_1=1.5 \\ \hdashline c_{21}=0 & d_{21}=0 & c_{22}=int_2=-1 & d_{22}=slp_2=1.5 \end{array}\right], \tag{23}$$

which in turn creates the following transition probability matrix.

$$\begin{array}{c} \\ Summer_{t_{i,j}} \\ Winter_{t_{i,j}} \end{array} \begin{array}{c} \begin{array}{cc} Summer_{t_{i,j+1}} & Winter_{t_{i,j+1}} \end{array} \\ \begin{pmatrix} \frac{exp(0+0\times cond)}{exp(0+0\times cond)+exp(int_1+slp_1\times cond)} & \frac{exp(int_1+slp_1\times cond)}{exp(0+0\times cond)+exp(int_1+slp_1\times cond)} \\ \frac{exp(0+0\times cond)}{exp(0+0\times cond)+exp(int_2+slp_2\times cond)} & \frac{exp(int_2+slp_2\times cond)}{exp(0)+exp(int_2+slp_2\times cond)} \end{pmatrix} \end{array} \tag{24}$$

Here we consider the Summer regime as the reference regime, so the first two columns of the transition LO matrix (Equation 23) are fixed at zero. The third and fourth columns of the transition LO matrix respectively correspond to the regression intercepts and slopes associated with the covariate, whose starting values are respectively set at -1 and 1.5. With this set of starting values, the transition probability from any regime to the Summer regime is .73 when `cond` $= 0$, and .38 when `cond` $= 1$. The negative intercept implies that in warmer days (`cond`

= 0), there is a greater chance of the process transitioning into the Summer regime, and the regression slope greater than the absolute value of the intercept suggests that in colder days (`cond` = 1), the transition into the Winter regime is more likely.

We fitted the specified model to the simulated data. Figure 4 is created by the `plotFormula()` method and presents the model equations with parameter names and estimated parameter values. Figure 5 is created from the `dynr.ggplot()` (or `autoplot()`) method with `style` set to 2, and shows that the predicted trajectories match with the observed values and alternate between different regimes. A complete modeling script for this example can be retrieved using `demo(RSNonlinearODE)`.

```
R> plotFormula(model2.2, ParameterAs = model2.2$param.names) +
+  ggtitle("(A)") +
+  theme(plot.title = element_text(hjust = 0.5, vjust = 0.01, size = 16))

R> plotFormula(model2.2, ParameterAs = coef(res2.2)) +
+  ggtitle("(B)") +
+  theme(plot.title = element_text(hjust = 0.5, vjust = 0.01, size = 16))

R> dynr.ggplot(res2.2, model2.2, style = 2,
+  names.regime = c("Summer", "Winter"),
+  title = "", idtoPlot = 9,
+  text = element_text(size = 16))
```

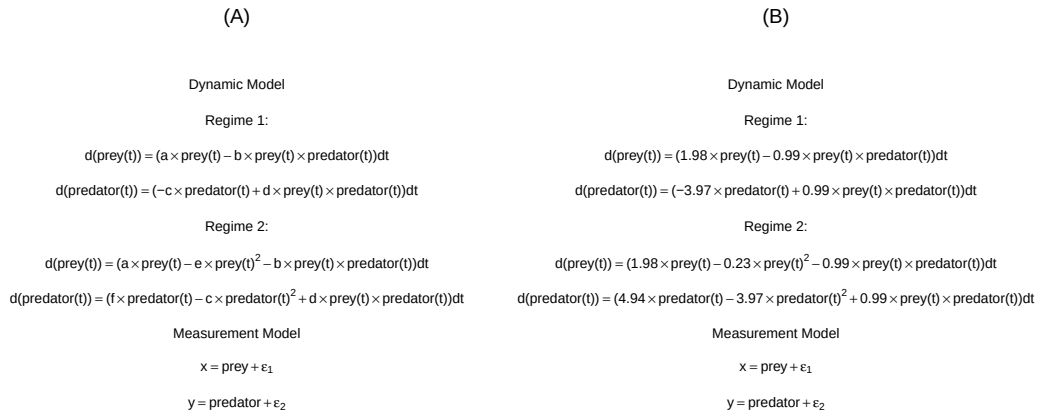

Figure 4: Automatic plots of model equations with (A) parameter names and (B) estimated parameters for the regime-switching nonlinear ODE model.

7. Other miscellaneous control options

In parameter estimation, `dynr` utilizes a sequential quadratic programming algorithm (Kraft 1988, 1994) available from an open-source library for nonlinear optimization — NLOPT (Johnson 2008). By default, we do not set boundaries on the parameters to be estimated. However,

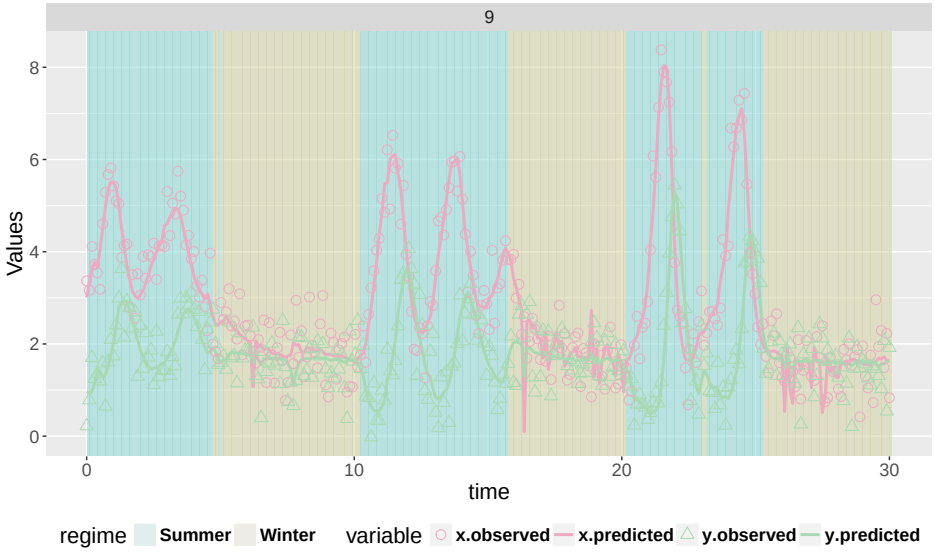

Figure 5: Built-in plotting feature for the predicted trajectories with observed values for the regime-switching nonlinear ODE model.

one can set the upper and lower boundaries of the estimated parameter values by respectively modifying the `ub` and `lb` slots of the model object of class `dynrModel`. An example is given as below to constrain the int_1 and int_2 parameters to be negative between -10 and 0, while limiting the values of slp_1 and slp_2 to positive within a range from 0 to 10:

```
R> model2.2$ub[ c("int_1", "int_2", "slp_1", "slp_2") ] <- c(0, 0, 10, 10)
R> model2.2$lb[ c("int_1", "int_2", "slp_1", "slp_2") ] <- c(-10, -10, 0, 0)
```

Similarly, the stopping criteria of the optimization algorithm can be modified through the `options` slot of the `dynrModel` object, which is a list consisting of specifications on the relative tolerance on optimization parameters (`xtol_rel`), the stopping threshold of the objective value (`stopval`), the absolute and relative tolerance on function value (i.e., `ftol_abs` and `ftol_rel`), the maximum number of function evaluations (`maxeval`), the maximum optimization time (in seconds; `maxtime`).

The output of the estimation function, `dynr.cook()`, is an object of class `dynrCook`. It not only includes estimation results that can be displayed in the summary table produced by `summary()`, but also contains information on posterior regime probabilities (i.e., the `pr_t_given_T` slot), smoothed state estimates of the latent variables (i.e., $\hat{\boldsymbol{\eta}}_i(t_{i,j}|T_i) = E(\boldsymbol{\eta}_i(t_{i,j})|\mathbf{Y}_i(T_i))$ in the `eta_smooth_final` slot), and smoothed error covariance matrices of the latent variables (i.e., $\mathbf{P}_i(t_{i,j}|T_i)$ in the `error_cov_smooth_final` slot) at all available time points. They can be retrieved by using the `$` operator.

8. Discussion and conclusions

This paper has introduced the **dynr** package. The increasing availability of ILD, particularly

in the social and behavioral sciences, has created a need for software specifically aimed at broad swaths of the social and behavioral science community. Such software require the capability of representing changes in latent variables over time in multiple people, possibly with discontinuous shifts in processes (i.e., regime switching). The interface for any package aimed at the broader community of researchers needs to carefully balance intuitive usability with a degree of flexibility in the specification to allow simple models to be written quickly while not limiting the user to only simple models. The **dynr** package attempts to meet these requirements and satisfy these needs.

The modeling framework embraced by **dynr** includes linear and nonlinear varieties of state-space modeling and differential equations as special cases. That is, **dynr** offers linear and nonlinear time series methods for latent variables in both the traditional discrete-time models and in the hybrid continuous-time models that have discrete measurements with continuous underlying processes. Additionally, regime-switching can be layered on top of any aspect of these models. To our knowledge, no other software allows for regime-switching nonlinear dynamics with latent variables. However, currently **dynr** only allows nonlinearity in the dynamics (i.e., in how the latent variables change over time) but not the measurement part of the model (i.e., in how the latent variables map onto the observed variables) to capitalize on the availability of a Gaussian approximate log-likelihood function for fast parameter estimation. Future extensions will incorporate Markov chain Monte Carlo (MCMC) techniques (e.g., Chow, Tang, Yuan, Song, and Zhu 2011; Durbin and Koopman 2001; Kim and Nelson 1999; Lu, Chow, Sherwood, and Zhu 2015) and pertinent frequentist-based estimation techniques (e.g., Fahrmeir and Tutz 1994) to accommodate a broader class of measurement models consisting of nonlinear functions and non-Gaussian densities.

Of course, numerous other programs exist for time series modeling, including ones for latent variable time series. Even though **dynr** can specify some models that these programs cannot, all of the features of these programs are not subsets of **dynr**. For example, **KFAS** allows for nonlinear measurement (Helske 2016) which is not currently possible in **dynr**. Moreover, SsfPack has nonlinear measurement capabilities along with many MCMC methods that **dynr** lacks (Koopman *et al.* 1999). The **pomp** package likewise has several features overlapping with **dynr**, but has also implemented several algorithms absent in **dynr**. **pomp** lists among its features hidden Markov models, state-space models, both of which can be discrete- or continuous-time, non-Gaussian, and nonlinear. **pomp** utilizes MCMC methods, Bayesian methods, particle filtering, as well as ensemble filtering and forecasting. However, **pomp** does not currently support regime-switching functionality beyond the regime switching found in hidden Markov modeling. Finally, facilities for multisubject designs are absent from these programs.

We have shown examples using **dynr** involving electromyography (EMG) data, and classic problems of predator-prey interaction. The EMG example made use of linear discrete-time modeling with regime-switching; whereas the predator-prey example included single-regime and regime-switching variations, both of which showcased the formula interface for creating nonlinear dynamics. These examples highlighted the use of *recipe* objects to prepare components of the model. The recipes divide the full model into meaningful conceptual chunks for ease of specification and interactive inspection. The recipes seamlessly handle various bookkeeping tasks like the creation and management of the free parameter vector and how free parameters map onto model components. This is in contrast to several other packages that require the user to do this management, often writing their own functions in the process.

In addition to sparing the user sundry bothersome tasks, the recipes allow for interactive error checking and model verification. The contents of each recipe can be printed in the R console, letting the user verify that the recipe they intended to specify was actually created. Along this vein, `plotFormula()` allows the user to see nicely formatted equations for their models directly in R, and `printex()` outputs LaTeX equations for their models which can be typeset immediately or modified for inclusion in manuscripts, presentations, and reports.

The cosmetic assets of the reporting features are complemented by remarkable speed in estimating models. All of the filtering code that produces the likelihood function for each model is written entirely in C, but this does not make **dynr** unique or give it computational advantage over many R packages. However, the optimization routine SLSQP (Kraft 1988, 1994) from the NLOPT (Johnson 2008) library keeps the entire likelihood evaluation *and* optimization in C. This strategy greatly reduces memory copying between R and C during optimization and does improve performance. Although a full performance comparison between packages is beyond the scope of the present work, early tests indicate that **dynr** is quite fast, readily estimating models with over a thousand rows of data in just a few seconds.

In addition to the need to accommodate a greater range of measurement functions/densities in the future, several other extensions are being pursued and implemented in the **dynr** package. For example, **dynr** currently handles missingness in the dependent variables via full-information maximum likelihood but does not allow for missingness in the covariates. Future plans include interfacing dynr with R packages such as **mice** (van Buuren and Groothuis-Oudshoorn 2011) to handle missingness in the covariates and/or dependent variables via multiple imputation. Further, models with nonlinearities at the dynamic level currently are not supported by well-established fit indices for evaluating and assessing model fit. While **dynr** provides the AIC (Akaike 1973) and the BIC (Schwarz 1978) for model comparison purposes. The tenability of using these model comparison criteria for comparing models involving nonlinearities at the dynamic level is yet to be investigated when the log-likelihood function to be optimized involves approximations and truncation errors. Finally, even though difference and differential equations have served as and remained one of the most popular modeling tools across myriad scientific disciplines, their use is still nascent in particular areas of the social and behavioral sciences. Tools to aid model developments and explorations (e.g., Chow, Bendezú, Cole, and Ram 2016; Ramsay, Hooker, and Graves 2009) are important extensions to enable and promote modeling efforts utilizing difference/differential equations. Fortunately, several existing packages in R offer many of the functionalities to support these modeling endeavors and may be used in conjunction or interfaced in the future with **dynr** for these purposes.

Even though dynamic models can be used to mathematically represent some of the theoretical postulates in social and behavioral sciences, the lack of readily accessible tools for evaluating more complex dynamic models contributes to the scarcity of modeling work along this line. In this paper, we provided two illustrative examples of the **dynr** package for analyzing time series data. By alleviating difficulties involved in the specification and estimation of the models, the dynamic modeling framework we present can serve as a valuable tool for evaluating substantive questions regarding human dynamic processes that are otherwise difficult to test within other frameworks. The **dynr** package allows users free access to computationally efficient algorithms from a simple and easy-to-learn interface for a broad class of linear and nonlinear, discrete- and continuous-time models. With robust, efficient, and accessible estimation algorithms, the application of dynamic models to ILD research will be more embraced and the quality of the

work will be enhanced. As a result, ILD research will become increasingly viable and continue to illuminate processes and correlates of change that can lead to evidence-based prevention and intervention programs that has the potential to change our lives.

References

Akaike H (1973). "Information Theory and an Extension of the Maximum Likelihood Principle." In BN Petrov, F Csaki (eds.), *Second International Symposium on Information Theory*, pp. 267–281. Akademiai Kiado, Budapest.

Anderson BDO, Moore JB (1979). *Optimal Filtering*. Prentice Hall, Englewood Cliffs, NJ.

Ansley CF, Kohn R (1985). "Estimation, Filtering and Smoothing in State Space Models with Incompletely Specified Initial Conditions." *The Annals of Statistics*, **13**, 1286–1316. doi:10.1214/aos/1176349739.

Bar-Shalom Y, Li XR, Kirubarajan T (2001). *Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software*. John Wiley & Sons, New York, NY.

Boker SM, Graham J (1998). "A Dynamical Systems Analysis of Adolescent Substance Abuse." *Multivariate Behavioral Research*, **33**(4), 479–507. doi:10.1207/s15327906mbr3304_3.

Boker SM, Neale MC, Klump KL (2014). "A Differenial Equations Model for the Ovarian Hormone Cycle." In PCM Molenaar, RM Lerner, KM Newell (eds.), *Handbook of Developmental Systems Theory and Methodology*, pp. 369–391. New York, NY.

Bolger N, Laurenceau JP (2013). *Intensive Longitudinal Methods: An Introduction to Diary and Experience Sampling Research*. Guilford Press, New York, NY.

Brown EN, Luithardt H (1999). "Statistical Model Building and Model Criticism for Human Circadian Data." *Journal of Biological Rhythms*, **14**, 609–616. doi:10.1177/074873099129000975.

Byrom B, Tiplady B (2010). *ePRO: Electronic Solutions for Patient-Reported Data*. Gower, Farnham, England.

Cacioppo JT, Petty R (1981). "Electromyograms as Measures of Extent and Affectivity of Information Processing." *American Psychologist*, **36**, 441–456. doi:10.1037//0003-066x.36.5.441.

Cacioppo JT, Petty RE, Losch ME, Kim HS (1986). "Electromyographic Activity over Facial Muscle Regions Can Differentiate the Valence and Intensity of Affective Reactions." *Journal of Personality and Social Psychology*, **50**(2), 260–268. doi:10.1037//0022-3514.50.2.260.

Chow SM, Bendezú JJ, Cole PM, Ram N (2016). "A Comparison of Two-Stage Approaches for Fitting Nonlinear Ordinary Differential Equation (ODE) Models with Mixed Effects." *Multivariate Behavioral Research*, **51**(2–3), 154–184. doi:10.1080/00273171.2015.1123138.

Chow SM, Ferrer E, Nesselroade JR (2007). "An Unscented Kalman Filter Approach to the Estimation of Nonlinear Dynamical Systems Models." *Multivariate Behavioral Research*, **42**(2), 283–321. doi:10.1080/00273170701360423.

Chow SM, Grimm KJ, Guillaume F, Dolan CV, McArdle JJ (2013). "Regime-Switching Bivariate Dual Change Score Model." *Multivariate Behavioral Research*, **48**(4), 463–502. doi:10.1080/00273171.2013.787870.

Chow SM, Ho MHR, Hamaker EJ, Dolan CV (2010). "Equivalences and Differences between Structural Equation and State-Space Modeling Frameworks." *Structural Equation Modeling*, **17**, 303–332. doi:10.1080/10705511003661553.

Chow SM, Nesselroade JR (2004). "General Slowing or Decreased Inhibition? Mathematical Models of Age Differences in Cognitive Functioning." *Journals of Gerontology B*, **59**(3), 101–109. doi:10.1093/geronb/59.3.P101.

Chow SM, Ou L, Ciptadi A, Prince E, Rehg JM, Rozga A, Messinger DS (Under review). "Differential Equation Modeling Approaches to Representing Sudden Shifts in Intensive Dyadic Interaction Data." *Psychometrika*.

Chow SM, Tang N, Yuan Y, Song X, Zhu H (2011). "Bayesian Estimation of Semiparametric Nonlinear Dynamic Factor Analysis Models using the Dirichlet Process Prior." *British Journal of Mathematical and Statistical Psychology*, **64**(1), 69–106. doi:10.1348/000711010x497262.

Chow SM, Witkiewitz K, Grasman RPPP, Maisto SA (2015). "The Cusp Catastrophe Model as Cross-Sectional and Longitudinal Mixture Structural Equation Models." *Psychological Methods*, **20**, 142–164. doi:10.1037/a0038962.

Chow SM, Zhang G (2013). "Nonlinear Regime-Switching State-Space (RSSS) Models." *Psychometrika*, **78**(4), 740–768. doi:10.1007/s11336-013-9330-8.

De Jong P (1988). "The Likelihood for a State Space Model." *Biometrika*, **75**(1), 165–169. doi:10.2307/2336450.

Dimberg U (1990). "Facial Electromyography and Emotional Reactions." *Psychophysiology*, **27**, 481–494. doi:10.1111/j.1469-8986.1990.tb01962.x.

Dimberg U, Thunberg M, Elmehed K (2000). "Unconscious Facial Reactions to Emotional Facial Expressions." *Psychological Science*, **11**(1), 86–89. doi:10.1111/1467-9280.00221.

Dolan CV (2005). "**MKFM6**: Multi-group, Multi-subject Stationary Time Series Modeling based on the Kalman Filter." URL http://users/fmg.uva.nl/cdolan/.

Dolan CV (2009). "Structural Equation Mixture Modeling." In RE Millsap, A Maydeu-Olivares (eds.), *The SAGE Handbook of Quantitative Methods in Psychology*, pp. 568–592. Sage, Thousand Oaks, CA.

Dolan CV, Jansen BR, Van der Maas HLJ (2004). "Constrained and Unconstrained Multivariate Normal Finite Mixture Modeling of Piagetian Data." *Multivariate Behavioral Research*, **39**(1), 69–98. doi:10.1207/s15327906mbr3901_3.

Driver CC, Oud JHL, Voelkle MC (in press). "Continuous Time Structural Equation Modelling with R Package **ctsem**." *Journal of Statistical Software*.

Durbin J, Koopman SJ (2001). *Time Series Analysis by State Space Methods*. Oxford University Press, Oxford, United Kingdom.

Fahrmeir L, Tutz G (1994). *Multivariate Statistical Modelling Based on Generalized Linear Models*. Springer–Verlag, New York, NY.

Fukuda K, Ishihara K (1997). "Development of Human Sleep and Wakefulness Rhythm during the First Six Months of Life: Discontinuous Changes at the 7th and 12th Week after Birth." *Biological Rhythm Research*, **28**, 94–103. doi:10.1076/brhm.28.3.5.94.13132.

Gilbert PD (2006 or later). *Brief User's Guide: Dynamic Systems Estimation*. URL http://cran.r-project.org/web/packages/dse/vignettes/Guide.pdf.

Gottman JM (2002). *The Mathematics of Marriage: Dynamic Nonlinear Models*. The MIT Press, Cambridge, MA.

Grayling MJ (2014). ***phaseR**: Phase Plane Analysis of One and Two Dimensional Autonomous ODE Systems*. R package version 1.3, URL https://CRAN.R-project.org/package=phaseR.

Grewal MS, Andrews AP (2008). *Kalman Filtering: Theory and Practice using MATLAB*. Third edition. John Wiley & Sons, Hoboken, NJ.

Hamilton JD (1989). "A New Approach to the Economic Analysis of Nonstationary Time Series and the Business Cycle." *Econometrica*, **57**, 357–384. doi:10.2307/1912559.

Hamilton JD (1994). *Time Series Analysis*. Princeton University Press, Princeton, NJ.

Harvey AC (1989). *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, Cambridge, United Kingdom.

Helske J (2016). "**KFAS**: Exponential Family State Space Models in R." *Accepted to Journal of Statistical Software*.

Hofbauer J, Sigmund K (1988). *The Theory of Evolution and Dynamical Systems: Mathematical Aspects of Selection (London Mathematical Society Student Texts)*. Cambridge University Press. ISBN 0521358388. URL http://www.worldcat.org/isbn/0521358388.

Hosenfeld B (1997). "Indicators of Discontinuous Change in the Development of Analogical Reasoning." *Journal of Experimental Child Psychology*, **64**, 367–395. doi:10.1006/jecp.1996.2351.

Hyndman RJ (2016). "CRAN Task View: Time Series Analysis." Online. Accessed on October 09, 2016., URL https://CRAN.R-project.org/view=TimeSeries.

Johnson SG (2008). *The NLopt Nonlinear-Optimization Package*. URL http://ab-initio.mit.edu/nlopt.

Kalman RE (1960). "A New Approach to Linear Filtering and Prediction Problems." *Journal of Basic Engineering*, **82**(1), 35–45. doi:10.1115/1.3662552.

Kim CJ, Nelson CR (1999). *State-Space Models with Regime Switching: Classical and Gibbs-Sampling Approaches with Applications*. MIT Press, Cambridge, MA.

Kohlberg L, Kramer R (1969). "Continuities and Discontinuities in Childhood and Adult Moral Development." *Human development*, **12**(2), 93–120. doi:10.1159/000270857.

Koopman SJ, Shephard N, Doornik JA (1999). "Statistical Algorithms for Models in State Space using **SsfPack** 2.2." *Econometrics Journal*, **2**(1), 113–166. doi:10.1111/1368-423X.00023.

Kraft D (1988). "A Software Package for Sequential Quadratic Programming." *Technical Report 88-28*, DFVLR-FB, Oberpfaffenhofen, Germany.

Kraft D (1994). "Algorithm 733: TOMP — Fortran Modules for Optimal Control Calculations." *ACM Transactions on Mathematical Software*, **20**(3), 262–281. doi:10.1145/192115.192124.

Kulikov GY, Kulikova MV (2014). "Accurate Numerical Implementation of the Continuous-Discrete Extended Kalman Filter." *IEEE Transactions on Automatic Control*, **59**(1). doi:10.1109/tac.2013.2272136.

Kulikova MV, Kulikov GY (2014). "Adaptive ODE Solvers in Extended Kalman Filtering Algorithms." *Journal of Computational and Applied Mathematics*, **262**, 205–216. doi:10.1016/j.cam.2013.09.064.

Lotka AJ (1925). *Elements of Physical Biology*. Williams & Wilkins, Baltimore, MD.

Lu ZH, Chow SM, Sherwood A, Zhu H (2015). "Bayesian Analysis of Ambulatory Cardiovascular Dynamics with Application to Irregularly Spaced Sparse Data." *Annals of Applied Statistics*, **9**, 1601–1620. doi:10.1214/15-AOAS846.

Muthén BO, Asparouhov T (2011). "LTA in Mplus: Transition Probabilities Influenced by Covariates." Mplus Web Notes: No. 13., URL http://www.statmodel.com/examples/{LTA}webnote.pdf.

Neale MC, Hunter MD, Pritikin JN, Zahery M, Brick TR, Kirkpatrick RM, Estabrook R, Bates TC, Maes HH, Boker SM (2016). "**OpenMx** 2.0: Extended Structural Equation and Statistical Modeling." *Psychometrika*, **80**(2), 535–549. doi:10.1007/s11336-014-9435-8.

Petris G (2010). "An R Package for Dynamic Linear Models." *Journal of Statistical Software*, **36**(12), 1–16. doi:10.18637/jss.v036.i12.

Petris G, Petrone S (2011). "State Space Models in R." *Journal of Statistical Software*, **41**(4), 1–25. doi:10.18637/jss.v041.i04.

Piaget J, Inhelder B (1969). *The Psychology of the Child*. Basic Books, New York, NY.

Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2002). *Numerical Recipes in C*. Cambridge University Press, Cambridge.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Ramsay JO, Hooker G, Graves S (2009). *Functional Data Analysis with R and MATLAB*. Springer–Verlag, New York, NY.

Rodgers JL, Rowe DC (1993). "Social Contagion and Adolescent Sexual Behavior: A Developmental EMOSA Model." *Psychological Review*, **100**(3), 479–510. doi:10.1037/0033-295X.100.3.479.

Rodgers JL, Rowe DC, Buster M (1998). "Social Contagion, Adolescent Sexual Behavior, and Pregnancy: a Nonlinear Dynamic EMOSA Model." *Developmental Psychology*, **34**(5), 1096–1113. doi:10.1037/0012-1649.34.5.1096.

Schnell S, Maini PK, Newman S, Newman TJ (eds.) (2008). *Multiscale Modeling of Developmental Systems. Series: Current Topics in Developmental Biology*. Academic Press., New York, NY.

Schwartz GE (1975). "Biofeedback, Self-regulation, and the Patterning of Physiological Processes." *American Scientist*, **63**, 314–324.

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**(2), 461–464. doi:10.1214/aos/1176344136.

Stone A, Shiffman S, Atienza A, Nebeling L (2008). *The Science of Real-Time Data Capture: Self-Reports in Health Research*. Oxford University Press, NY.

Thatcher RW (1998). "A Predator-Prey Model of Human Cerebral Development." In KM Newell, PCM Molenaar (eds.), *Applications of Nonlinear Dynamics to Developmental Process Modeling*, pp. 87–128. Lawrence Erlbaum, Mahwah, NJ.

The MathWorks, Inc (2016). *MATLAB version 9.1 (R2016b)*. The MathWorks, Inc., Natick, MA.

Thomas EA, Martin JA (1976). "Analyses of Parent-Infant Interaction." *Psychological Review*, **83**(2), 141–156. doi:10.1037/0033-295X.83.2.141.

Tiao GC, Tsay RS (1994). "Some Advances in Non-Linear and Adaptive Modelling in Time Series." *Journal of Forecasting*, **13**, 109–131. doi:10.1002/for.3980130206.

Tong H, Lim KS (1980). "Threshold Autoregression, Limit Cycles and Cyclical Data." *Journal of the Royal Statistical Society B*, **42**, 245–292. doi:10.1142/9789812836281_0002.

van Buuren S, Groothuis-Oudshoorn K (2011). "**mice**: Multivariate Imputation by Chained Equations in R." *Journal of Statistical Software*, **45**(3), 1–67. doi:10.18637/jss.v045.i03.

van der Maas HLJ, Kolstein R, van der Pligt J (2003). "Sudden Transitions in Attitudes." *Sociological Methods & Research*, **32**(125–152). doi:10.1177/0049124103253773.

van der Maas HLJ, Molenaar PCM (1992). "Stagewise Cognitive Development: An Application of Catastrophe Theory." *Psychological Review*, **99**(3), 395–417. doi:10.1037//0033-295x.99.3.395.

van Dijk M, van Geert P (2007). "Wobbles, Humps and Sudden Jumps: A Case Study of Continuity, Discontinuity and Variability in Early Language Development." *Infant and Child Development*, **16**(1), 7–33. doi:10.1002/icd.506.

Volterra V (1926). "Fluctuations in the Abundance of a Species considered Mathematically." *Nature*, **118**, 558–560. doi:10.1038/118558a0.

Yang M, Chow SM (2010). "Using State-Space Model with Regime Switching to Represent the Dynamics of Facial Electromyography (EMG) Data." *Psychometrika: Application and Case Studies*, **74**(4), 744–771. doi:10.1007/s11336-010-9176-2.

Affiliation:

Funding for this study was provided by NSF grant SES-1357666, NIH grants R01MH61388, R01HD07699, R01GM105004, Pennsylvania State Quantitative Social Sciences Initiative, and UL TR000127 from the National Center for Advancing Translational Sciences.

Lu Ou
Department of Human Development and Family Studies
The Pennsylvania State University
420 Biobehavioral Health Building
University Park, PA 16802
E-mail: lzo114@psu.edu

Michael D. Hunter
Center on Child Abuse and Neglect
University of Oklahoma Health Sciences Center
940 NE 13th St, Suite 4900
Oklahoma City, OK 73104
E-mail: mhunter1@ouhsc.edu