Example: Regime-switching Linear Discrete-time Model

Lu Ou, Michael D. Hunter, Sy-Miin Chow

April 1, 2019

This file demonstrates the utilization of **dynr** in fitting a regime-switching linear dynamic models. A complete modeling script for this example is available as one of the demo examples in **dynr** and can be found using

```
> file.edit(system.file("demo", "RSLinearDiscreteYang.R", package = "dynr"))
```

1 Regime-switching State Space Model

Facial electromyography (EMG) has been used in the behavioral sciences as one possible indicator of human emotions [**Schwartz75a**, **Cacioppo81a**, **Cacioppo86a**, **Dimberg00a**]. 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 [**Schwartz75a**, **Dimberg90a**, **Cacioppo81a**].

A time series of EMG data contains bursts of electrical activity that are typically magnified when an individual is under emotion induction. [**Yang10a**] proposed using a regime-switching linear state-space model in which the individual may transition between regimes with and without facial EMG activation. As such, 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**.

The model of interest is the final model selected for this participant by [**Yang10a**]:

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

$$\eta_i(t_{i,j+1}) = \phi_{S_i(t_{i,j})}\eta_i(t_{i,j}) + \zeta_i(t_{i,j+1}), \tag{2}$$

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.

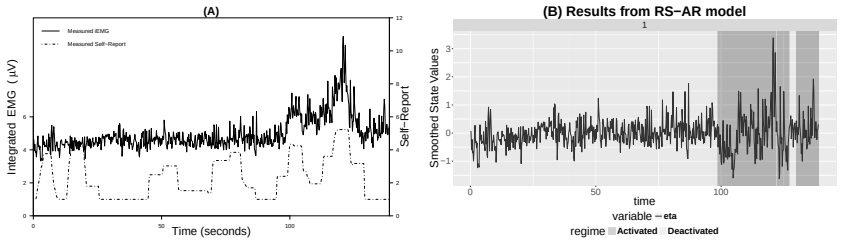

Figure 1: (A) A plot of integrated electromyography (iEMG) and self–report affect ratings for one participant with a time interval of 0.2 seconds between two adjacent observations. Self–report = self–report affect ratings; iEMG = integrated EMG signals. (B) An automatic plot of the smoothed state estimates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

1.1 Prepare the data

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

```
> #---- Load packages ----
> require("dynr")
> #---- Read in data and create dynr data object----
> data("EMG")
> 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 measurement 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.

1.2 Prepare the recipes

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.

1.2.1 Model specification: the dynamic functions

The dynamic model can take on the form of continuous-time models as

$$d\boldsymbol{\eta}_i(t) = \boldsymbol{f}_{S_i(t)}\left(\boldsymbol{\eta}_i(t), t, \boldsymbol{x}_i(t)\right) dt + d\boldsymbol{w}_i(t), \tag{3}$$

or the form of discrete-time state-space models [**Durbin01a**] as

$$\boldsymbol{\eta_i}(t_{i,j+1}) = \boldsymbol{f}_{S_i(t)}\left(\boldsymbol{\eta_i}(t_{i,j}), t_{i,j}, \boldsymbol{x}_i(t_{i,j})\right) + \boldsymbol{w}_i(t_{i,j+1}), \tag{4}$$

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. $\boldsymbol{f}_{S_i(t)}(.)$ depends on the latent regime indicator, $S_i(t)$, the discrete-valued latent variable that indexes the operating regime at time t.

The dynamic functions, $\boldsymbol{f}_{S_i(t)}()$ in Equations 3 and 4, 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 4 reduces to:

$$\boldsymbol{\eta_i}(t_{i,j+1}) = \boldsymbol{\alpha}_{S_i(t_{i,j})} + \boldsymbol{F}_{S_i(t_{i,j})}\boldsymbol{\eta_i}(t_{i,j}) + \boldsymbol{B}_{S_i(t_{i,j})}\boldsymbol{x}_i(t_{i,j}) + \boldsymbol{w}_i(t_{i,j+1}), \tag{5}$$

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 3.

$$d\boldsymbol{\eta_i}(t) = \left(\boldsymbol{\alpha}_{S_i(t)} + \boldsymbol{F}_{S_i(t)}\boldsymbol{\eta_i}(t) + \boldsymbol{B}_{S_i(t)}\boldsymbol{x}_i(t)\right) dt + d\boldsymbol{w}_i(t), \tag{6}$$

In this example, the dynamics as in Equation 2 are linear and discrete-time, so we can describe the dynamics in terms of Equation 5 as

$$\eta_i(t_{i,j+1}) = \underbrace{0}_{\boldsymbol{\alpha}_{S_i(t_{i,j})}} + \underbrace{\phi_{S_i(t_{i,j})}}_{\boldsymbol{F}_{S_i(t_{i,j})}} \eta_i(t_{i,j}) + \underbrace{0}_{\boldsymbol{B}_{S_i(t_{i,j})}} \boldsymbol{x}_i(t_{i,j}) + \underbrace{\zeta_i(t_{i,j+1})}_{\boldsymbol{w}_i(t_{i,j+1})}. \tag{7}$$

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 2. 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 2. 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.

```
> #---- Dynamic functions ----
> 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)
```

1.2.2 Model specification: the linear measurement function

For both discrete- and continuous-time models, we assume that 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, $\boldsymbol{y}_i(t_{i,j})$ as

$$\boldsymbol{y}_i(t_{i,j}) = \boldsymbol{\tau}_{S_i(t_{i,j})} + \boldsymbol{\Lambda}_{S_i(t_{i,j})}\boldsymbol{\eta}_i(t_{i,j}) + \mathbf{A}_{S_i(t_{i,j})}\boldsymbol{x}_i(t_{i,j}) + \boldsymbol{\epsilon}_i(t_{i,j}), \quad \boldsymbol{\epsilon}_i(t_{i,j}) \sim N\left(\mathbf{0}, \mathbf{R}_{S_i(t_{i,j})}\right), \tag{8}$$

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, $\mathbf{R}_{S_i(t_{i,j})}$.

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

1.2.3 Model specification: the latent and observed noise covariance structures

The noise recipe is created with *prep.noise()*. $\boldsymbol{w}_i(t)$ in Equation 3 is 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. In Equation 4, however, $\boldsymbol{w}_i(t)$ denotes a vector of Gaussian distributed process noise with regime-specific covariance matrix, $\mathbf{Q}_{S_i(t)}$. In both continuous- and discrete-time models, $\mathbf{Q}_{S_i(t)}$ can be specified by the **.latent* arguments in *prep.noise()*. The **.observed* arguments are for $\mathbf{R}_{S_i(t_{i,j})}$ in Equation 8.

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.

```
> #---- Dynamic and measurement noise cov structures----
> 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))
```

1.2.4 Model specification: the initial condition

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, $\boldsymbol{\Sigma}_{\boldsymbol{\eta}_1}$:

$$\boldsymbol{\eta}_i(t_{i,1}) \sim N\left(\boldsymbol{\mu}_{\boldsymbol{\eta}_1}, \boldsymbol{\Sigma}_{\boldsymbol{\eta}_1}\right). \tag{9}$$

The subscript $S_i(t)$ that appears in Equations 3–8 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

$$\Pr\left(S_i(t_{i,1}) = m | \boldsymbol{x}_i(t_{i,1})\right) \quad \stackrel{\Delta}{=} \pi_{m,i1} = \frac{\exp(a_m + \mathbf{b}_m^T \boldsymbol{x}_i(t_{i,1}))}{\sum_{k=1}^{M} \exp(a_k + \mathbf{b}_k^T \boldsymbol{x}_i(t_{i,1}))}, \tag{10}$$

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 10 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.

The *prep.initial()* function is used to specify the $\boldsymbol{\mu}_{\boldsymbol{\eta}_1}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\eta}_1}$ in Equation 9, as well as the model for the initial regime probabilities (i.e., Equation 10).

```
> #---- Initial condition specification ----
> recIni <- prep.initial(
+   values.inistate = matrix(0, 1, 1),
+   params.inistate = matrix('fixed', 1, 1),
+   values.inicov = matrix(1, 1, 1),
+   params.inicov = matrix('fixed', 1, 1),
+   values.regimep = c(1, 0),
+   params.regimep = c('fixed', 'fixed'))
```

1.2.5 Model specification: the regime-switching model

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 $t_{i,j}$. The rows of this matrix sum to 1.0 because the probability of transitioning from a particular state to any other state must be 1.0. 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:

$$\Pr\left(S_i(t_{i,j}) = m | S_i(t_{i,j-1}) = l, \boldsymbol{x}_i(t_{i,j})\right) \quad \triangleq \pi_{lm,it} = \frac{\exp(c_{lm} + \mathbf{d}_{lm}^T \boldsymbol{x}_i(t_{i,j})}{\sum_{k=1}^{M} \exp(c_{lk} + \mathbf{d}_{lk}^T \boldsymbol{x}_i(t_{i,j}))} \quad (11)$$

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., $\sum_{m=1}^{M} \pi_{lm} = 1$).

The *prep.regimes()* function specifies the structure of the regime switching functions shown in Equation 11. Note that based on Equation 11, 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:

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

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.

```
> # ---- Regimes-switching model ----
> 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:

$$\begin{array}{c} \\ Deactivated_{t_{i,j}} \\ Activated_{t_{i,j}} \end{array} \begin{array}{c} \begin{array}{cc} Deactivated_{t_{i,j+1}} & Activated_{t_{i,j+1}} \end{array} \\ \begin{pmatrix} \frac{exp(c_{11})}{exp(c_{11})+exp(0)} & \frac{exp(0)}{exp(c_{11})+exp(0)} \\ \frac{exp(c_{21})}{exp(c_{21})+exp(0)} & \frac{exp(0)}{exp(c_{21})+exp(0)} \end{pmatrix} \end{array} \tag{13}$$

with starting values of

$$\begin{array}{c} \\ Deactivated_{t_{i,j}} \\ Activated_{t_{i,j}} \end{array} \begin{array}{c} \begin{array}{cc} Deactivated_{t_{i,j+1}} & Activated_{t_{i,j+1}} \end{array} \\ \begin{pmatrix} .668 & .332 \\ .269 & .731 \end{pmatrix}. \end{array} \tag{14}$$

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:

$$c_{lm} = c_m + c_{\Delta,lm} \tag{15}$$

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 11 now appears as:

$$\Pr\left(S_i(t_{i,j}) = m | S_i(t_{i,j-1}) = l, \boldsymbol{x}_i(t_{i,j})\right) \quad \stackrel{\Delta}{=} \pi_{lm,it} = \frac{\exp(c_m + c_{\Delta,lm} + \mathbf{d}_{lm}^T \boldsymbol{x}_i(t_{i,j})}{\sum_{k=1}^{M} \exp(c_k + c_{\Delta,lk} + \mathbf{d}_{lk}^T \boldsymbol{x}_i(t_{i,j}))} \tag{16}$$

and all parameters in Equation 16 can be summarized into

$$\left[\begin{array}{cc:cc:c:cc} c_{\Delta,11} & \mathbf{d}_{11}^T & c_{\Delta,12} & \mathbf{d}_{12}^T & \cdots & c_{\Delta,1M} & \mathbf{d}_{1M}^T \\ \hdashline c_{\Delta,21} & \mathbf{d}_{21}^T & c_{\Delta,22} & \mathbf{d}_{22}^T & \cdots & c_{\Delta,2M} & \mathbf{d}_{2M}^T \\ \hdashline \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hdashline c_1 & \mathbf{d}_{M1}^T & c_2 & \mathbf{d}_{M2}^T & \cdots & c_M & \mathbf{d}_{MM}^T \end{array}\right], \tag{17}$$

if we use regime M for the reference row and hence have $c_{\Delta,Ml} = 0$ for $l = 1$, ..., M. This allows the same parameter matrix structure for both the deviation form (Equation 17) and non-deviation form (Equation 12) 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 16 and 11) are available in **dynr**. 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 $l = 1$, ..., 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. Users can specify regime-switching log-odds in deviation form by invoking the optional argument, *deviation=TRUE* in the call to *prep.regimes*.

```
> 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 14 but parameterized in the form of Equation 17. The deviation form can be extremely useful for testing hypotheses about the relationships between LO intercepts and for making constraints across regimes.

1.3 Create and cook the model

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.

```
> #---- Create model  ----
>
> rsmod <- dynr.model(
+   dynamics = recDyn,
+   measurement = recMeas,
+   noise = recNoise,
+   initial = recIni,
+   regimes = recReg,
+   data = EMGdata,
+   outfile = "RSLinearDiscrete.c")
> #---- Create model and cook it all up  ----
>
> 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**.

1.4 Serve the results

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.

```
> #---- Serve it! ----
> summary(yum)
```

These parameter estimates, standard errors, and likelihood values closely mirror those reported in [**Yang10a**]. 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

$$\begin{array}{c} \\ Deactivated_{t_{i,j}} \\ Activated_{t_{i,j}} \end{array} \begin{array}{c} \begin{array}{cc} Deactivated_{t_{i,j+1}} & Activated_{t_{i,j+1}} \end{array} \\ \begin{pmatrix} .9959 & .0041 \\ .0057 & .9943 \end{pmatrix} \end{array} \tag{18}$$

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()*.

```
> 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).

```
> #pdf('./Figures/plotRSGG.pdf', height=7, width=12)
> 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 = 24),
+   is.bw = TRUE)
> #dev.off()
>
> autoplot(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 latent state $\eta_i(t_{i,j+1})$ varying according to the lower autocorrelation 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 latent state becomes 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. Of course, note that Figure 1(A) shows the observed EMG data whereas Figure 1(B) shows the latent state which is related to the observed data by Equation 1.

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 *names(rsmod)*, 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.

```
> plotFormula(dynrModel = rsmod, ParameterAs = names(rsmod),
+   printDyn = TRUE, printMeas = TRUE) +
+   ggtitle("(A)") +
```

```
+   theme(plot.title = element_text(hjust = 0.5, vjust = 0.01, size = 16))
> 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 1 and 2 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.

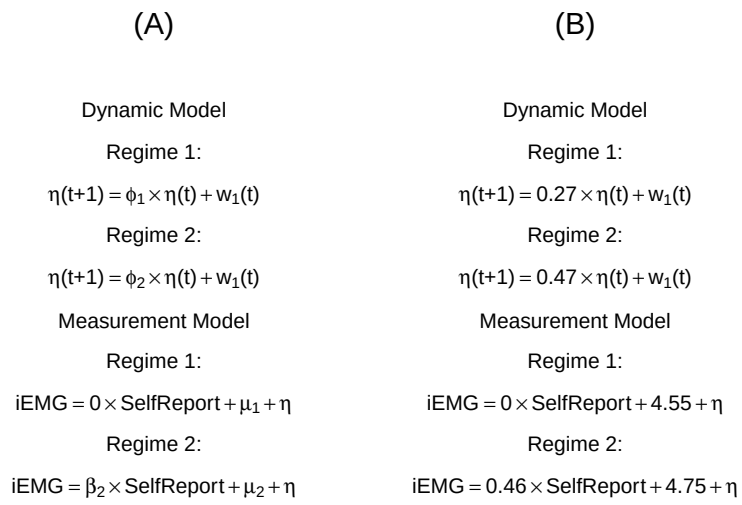

Figure 2: Automatic plots of model equations with (A) parameter names and (B) estimated parameters for the regime-switching linear state-space model.

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

```
> printex(rsmod,
+   ParameterAs = names(rsmod),
+   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 *names(rsmod)* 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, "RSLinearDiscreteYang.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:

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

This example has used real EMG data from a previous study [**Yang10a**] to illustrate many parts of the user-interface for **dynr**. 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.