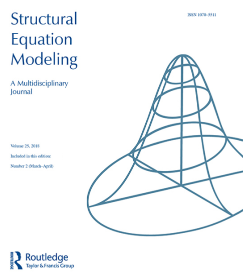

Structural Equation Modeling: A Multidisciplinary Journal

ISSN: 1070-5511 (Print) 1532-8007 (Online) Journal homepage: http://www.tandfonline.com/loi/hsem20

State Space Modeling in an Open Source, Modular, Structural Equation Modeling Environment

Michael D. Hunter

To cite this article: Michael D. Hunter (2018) State Space Modeling in an Open Source, Modular, Structural Equation Modeling Environment, Structural Equation Modeling: A Multidisciplinary Journal, 25:2, 307-324, DOI: 10.1080/10705511.2017.1369354

To link to this article: https://doi.org/10.1080/10705511.2017.1369354

Published online: 23 Oct 2017.

Submit your article to this journal

Article views: 83

View related articles

View Crossmark data

Full Terms & Conditions of access and use can be found at
http://www.tandfonline.com/action/journalInformation?journalCode=hsem20

Structural Equation Modeling: A Multidisciplinary Journal, 25: 307–324, 2018
Copyright © Taylor & Francis Group, LLC
ISSN: 1070-5511 print / 1532-8007 online
DOI: https://doi.org/10.1080/10705511.2017.1369354

State Space Modeling in an Open Source, Modular, Structural Equation Modeling Environment

Michael D. Hunter
University of Oklahoma Health Sciences Center

State space models (SSMs) are introduced in the context of structural equation modeling (SEM). In particular, the OpenMx implementation of SSMs using the Kalman filter and prediction error decomposition is discussed. In reflection of modularity, the implementation uses the same full information maximum likelihood missing data procedures for SSMs and SEMs. Similarly, generic OpenMx features such as likelihood ratio tests, profile likelihood confidence intervals, Hessian-based standard errors, definition variables, and the matrix algebra interface are all supported. Example scripts for specification of autoregressive models, multiple lag models (VAR(p)), multiple lag moving average models (VARMA(p, q)), multiple subject models, and latent growth models are provided. Additionally, latent variable calculation based on the Kalman filter and raw data generation based on a model are all included. Finally, future work for extending SSMs to allow for random effects and for presenting them in diagrams is discussed.

Keywords: State space model, Software, Kalman filter, OpenMx

For years, psychologists and other behavioral researchers have used structural equation models (SEMs) to describe the between-person structure of human actions. In many cases, these SEMs are static, corresponding to a single measurement occasion. Single-occasion models are intrinsically limited in their applicability to within-person processes unless some strong assumptions are invoked (see Bollen & Curran, 2006; Hamaker, Dolan, & Molenaar, 2005, for details on these assumptions and the likelihood of them being met). However, several branches of research have extended SEMs in various ways to explore longitudinal processes in the SEM context. Latent growth curve models (McArdle & Epstein, 1987; Molenaar, 2004; Muthén & Curran, 1997), panel designs (Boker, Neale, & Rausch, 2004; Boker & Nesselroade, 2002; Bollen & Curran, 2004; Duncan, 1969), and dynamic factor analyses (Browne & Nesselroade, 2005; Molenaar, 1985) are examples of such extended SEMs. These SEM extensions have improved the representation of within-person processes greatly. Nonetheless, when the numbers of time points and variables increase in more intensive longitudinal designs, these discrete extensions are cumbersome to implement and inadequate to describe longitudinal change. In those cases, state space models (SSMs; Anderson, 1963; Hamaker, Kuiper, & Grasman, 2015; Hamerle, Nagl, & Singer, 1991; Voelkle & Oud, 2015) instead provide a more flexible modeling paradigm for describing this type of longitudinal change.

The way that SSMs represent change generally differs from that in SEMs. When modeling in discrete time, SSMs are a recursive representation of a process: The current state is built on the previous state. By contrast, latent growth models, as a prototypical SEM example, are an explicit representation of the whole change process at once. Both the recursive and the explicit forms of a process can be valuable. Sometimes the recursive form is simpler, is more general, or reveals underlying mechanisms that are occluded by the explicit form. Take, for example, the Fibonacci sequence which has as its first few terms 0, 1, 1, 2, 3, 5, 8, 13, and so on. There are both explicit and recursive formulas for this sequence. The explicit formula has

Correspondence should be addressed to Michael D. Hunter, Department of Pediatrics, University of Oklahoma Health Sciences Center, Oklahoma City, OK 73104, USA. E-mail: mhunter.ou@gmail.com

Supplemental data for this article can be accessed at www.tandfonline.com/hsem.

$$x_n = \frac{a^n - \left(\frac{-1}{a}\right)^n}{a - \left(\frac{-1}{a}\right)} \text{ with } a = 1.61803\ldots \quad (1)$$

whereas the recursive formula has

$$x_n = x_{n-1} + x_{n-2} \text{ with } x_0 = 0 \text{ and } x_1 = 1. \quad (2)$$

Of these two, the recursive statement is by far easier to understand. Of course, it is not always the case that the recursive form of a change process is easier to grasp, but it does more directly relate to the conception of change as an accumulation over time. Thus in the recursive conception, change is a process that builds on its history.

In this article, we first introduce SSMs by way of their relation to SEMs. Second, we describe a distinct notation for SSMs. Third, we provide further computational detail for SSMs and their particular method of representing change over time. Fourth, we discuss the modular implementation of SSMs in the R package OpenMx (Neale et al., 2016), including several features that are not typically available. Finally, we show several common examples of SSMs with complete code illustrating features of the SSM module.

A RELATION BETWEEN STRUCTURAL EQUATION AND STATE SPACE MODELS

SSMs are the latent analogue to autoregressive and autoregressive moving average (ARMA) models. Because the variables of interest are not directly observed, these models are more complex. SSMs are a form of latent variable model in which the latent variables for one row of data are not independent from those at another row of data. The dependence is typically written in terms of a lag-one vector autoregression. That is, the vector of latent variables at one time point predicts those at the next time point, but not those two or more time points ahead. Thus, the dependence is first-order Markov. However, it can be shown that any higher order lag autoregressions and arbitrary-order moving averages can be incorporated into a first-order Markov dependence structure (e.g., Hamilton, 1994, pp. 3043–3044). Hence, although the typical form of an SSM appears to be strictly lag-one autoregressive, in reality any vector autoregressive moving average (VARMA) model can be specified in this form. Therefore, SSMs are a superset of VARMA models. The primary distinction between SSMs and VARMA models is that SSMs also allow for latent variables, whereas VARMA acts directly on the observed variables, ignoring the possibility of measurement error and the other benefits that latent variables provide. So, SSMs are multivariate time series models in which the time series is a latent process, not an observed one.

Because this article describes the implementation of SSMs in a program originally designed only for SEMs, we find it helpful to describe SSMs as a variation of SEMs, even though their developmental histories are almost completely independent. It is fruitful to think of SSMs as a kind of SEM that specifically handles time. It will be shown in this article that every SEM can be represented as an SSM with no dynamics and a special latent covariance structure. The reverse that every SSM can be represented as a blockwise constructed SEM has been shown previously (MacCallum & Ashby, 1986). In other words, both techniques are sufficiently flexible to encompass one another. The only remaining considerations are the computational efficiency and ease of specification for the model under investigation. For particular kinds of models, it might be easier, more natural, or more computationally expedient to specify it in one form or another. Generally speaking, models that involve time appear simpler as SSMs than SEMs; models that do not involve time most often seem simpler as SEMs.

In the conventional M*plus*/LISCOMP (Muthén & Satorra, 1995; Muthén & Muthén, 1998–2014) notation for SEMs, a model is specified by the matrices

$$\boldsymbol{\eta}_i = \mathbf{B}\boldsymbol{\eta}_i + \boldsymbol{\Gamma}\mathbf{x}_i + \zeta_i \quad (3)$$

$$\mathbf{y}_i = \boldsymbol{\Lambda}\boldsymbol{\eta}_i + \mathbf{K}\mathbf{x}_i + \varepsilon_i \quad (4)$$

where $\boldsymbol{\eta}_i$ is the vector of latent variables for row i, $\mathbf{B}$ is the matrix of regression coefficients between latent variables, $\mathbf{x}_i$ is the vector of exogenous observed variables (covariates) with regression effects given in $\boldsymbol{\Gamma}$, ζ_i are the latent residual effects with covariance $\boldsymbol{\Psi}$, $\boldsymbol{\Lambda}$ gives the factor loadings, $\boldsymbol{K}$ gives the effects of $\mathbf{x}_i$ on the observed variables $\mathbf{y}_i$, and the manifest residuals are $\mathbf{e}_i$ with covariance $\boldsymbol{\Theta}$. Note that the intercept terms ($\boldsymbol{\alpha}$ and $\mathbf{v}$), which are sometimes written with the SEM can always be incorporated as part of the covariate model (matrices $\boldsymbol{\Gamma}$ and $\boldsymbol{K}$) by having one of the exogenous variables in $\boldsymbol{x}$ be the constant 1. The analogous equations in the same notation for SSMs are

$$\boldsymbol{\eta}_i = \mathbf{B}\boldsymbol{\eta}_{i-1} + \boldsymbol{\Gamma}\mathbf{x}_i + \zeta_i \quad (5)$$

$$\mathbf{y}_i = \boldsymbol{\Lambda}\boldsymbol{\eta}_i + \mathbf{K}\mathbf{x}_i + \varepsilon_i \quad (6)$$

The difference between these two sets of equations is subtle and difficult to see at first. There is only one difference. In the right side of Equation 3, the subscript on $\boldsymbol{\eta}$ differs from that on the right side of Equation 5. Equations 4 and 6 are identical. Put another way, the measurement models of SEM and SSM are the same; their structural models differ slightly, but importantly.

The shift in the subscript of $\boldsymbol{\eta}$ allows the latent variables at one measurement occasion to relate to those at a separate measurement occasion. Traditionally, the subscripted unit i is considered time, but the same autoregressive model structure can be exploited for spatial analyses (Holmes, Ward, & Wills, 2012) and other forms of data clustering (Gu, Preacher, Wu, & Yung, 2014). The same dependence across units is possible in SEMs, but requires all dependent units to be collected together in the same latent variable vector. Most often, this is done with block matrices. For example, by defining

$$\boldsymbol{\eta}_i = \begin{pmatrix} \boldsymbol{\eta}_{i,1} \\ \boldsymbol{\eta}_{i,2} \end{pmatrix}; \; \boldsymbol{B} = \begin{pmatrix} \boldsymbol{B}_{1,1} & \boldsymbol{B}_{1,2} \\ \boldsymbol{B}_{2,1} & \boldsymbol{B}_{2,2} \end{pmatrix}; \; \textit{et cetera}; \tag{7}$$

multiple latent variable vectors can be gathered into a single block-wise latent variable model. In theory, this allows SEMs to represent autoregressive, dynamic, and growth processes without recourse to SSMs. However, in practice the latent dimension needed for these time-oriented models becomes prohibitively large very quickly. For example, a lag-one autoregressive model for $T = 100$ time points on $p = 9$ variables requires $Tp = 900$ variables to be represented as an SEM. Hence, the expected covariance matrix for such a model is 900×900 and requires considerable time to repeatedly invert for the likelihood function evaluation. The same model in SSM form requires only $p = 9$ variables. Its 9×9 covariance matrix is much faster to invert, and much simpler to specify. Thus, time-oriented models can be specified as SEMs. At some point, though, the cost–benefit tips in the favor of directly using a time-oriented modeling framework like SSMs.

To avoid notational confusion between SSMs and SEMs, this article uses a notation for SSMs typically found in engineering. This is also the notation used in OpenMx (Neale et al., 2016). In this notation, an SSM is written

$$\boldsymbol{x}_t = \boldsymbol{A}\boldsymbol{x}_{t-1} + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{q}_t \tag{8}$$

$$\boldsymbol{y}_t = \boldsymbol{C}\boldsymbol{x}_t + \boldsymbol{D}\boldsymbol{u}_t + \boldsymbol{r}_t \tag{9}$$

where $\boldsymbol{x}_t$ is an $l \times 1$ vector of the latent states at time or other index t, $\boldsymbol{u}_t$ is an $m \times 1$ vector of observed inputs (or covariates or exogenous variables), $\boldsymbol{q}_t$ is an $l \times 1$ vector of dynamic noise with mean zero and covariance $\boldsymbol{Q}$, $\boldsymbol{y}_t$ is an $n \times 1$ vector of observed outputs (or manifest variables), $\boldsymbol{r}_t$ is an $n \times 1$ vector of observation noise with mean zero and covariance $\boldsymbol{R}$, $\boldsymbol{A}$ is an $l \times 1$ matrix of autoregressive dynamics, $\boldsymbol{B}$ is an $l \times m$ matrix of covariate/input effects on the state, $\boldsymbol{C}$ is an $n \times l$ matrix of factor loadings, and $\boldsymbol{D}$ is an $n \times m$ matrix of covariate/input effects on the observation.

Equation 8 is called the state equation; it is analogous to the structural model in structural equation models. Equation 9 is called the output equation; it is identical to the measurement model in SEMs. The noise vectors $\boldsymbol{q}_t$ and $\boldsymbol{r}_t$ have zero mean, are assumed to be uncorrelated with each other, uncorrelated with themselves at other times, and uncorrelated with the observations $\boldsymbol{y}_t$. In mathematical notation,

$$\begin{aligned} E[\boldsymbol{r}_t] = \mathbf{0}_{n\times 1}; \quad E[\boldsymbol{r}_t\boldsymbol{r}_s^T] = \delta(t-s)\mathbf{R}; \\ E[\mathbf{q}_t] = \mathbf{0}_{l\times 1}; \quad E[\boldsymbol{q}_t\mathbf{q}_s^T] = \delta(t-s)\mathbf{Q}; \end{aligned} \tag{10}$$

$$\begin{aligned} E[\mathbf{r}_t\mathbf{q}_s^T] = \mathbf{0}_{n\times l}; \quad E[\boldsymbol{y}_t\mathbf{q}_s^T] = \mathbf{0}_{n\times l}; \\ E[\boldsymbol{y}_t\mathbf{r}_s^T] = \mathbf{0}_{n\times n} \end{aligned} \tag{11}$$

where $E[\,]$ is the expectation operator, ^{T} is the transpose function, and $\delta()$ is the Dirac delta function (unity at zero and zero everywhere else).

As mentioned previously, it can be shown that every SEM can be specified as an SSM with zero dynamics. We have not seen this demonstrated in the literature, so it is included here. First, note that the structural equation model (Equation 3) is equivalent to

$$\boldsymbol{\eta}_i = (\boldsymbol{I} - \boldsymbol{B})^{-1}\boldsymbol{\Gamma}\mathbf{x}_i + \zeta'_i \tag{12}$$

where ζ'_i defines a new residual vector of the same dimension and mean as ζ_i but with $\mathrm{Cov}(\zeta'_i) = (\boldsymbol{I} - \boldsymbol{B})^{-1}\Psi(\boldsymbol{I} - \boldsymbol{B})^{-\mathrm{T}}$. Because of this, there is always an SSM representation of the SEM. This representation has

$$\boldsymbol{x}_i = \boldsymbol{\eta}_i; \quad \boldsymbol{y}_i = \boldsymbol{y}_i; \quad \boldsymbol{u}_i = \boldsymbol{x}_i; \tag{13}$$

$$A = \mathbf{0}_{l\times l}; \quad \boldsymbol{B} = (\boldsymbol{I} - \boldsymbol{B})^{-1}\boldsymbol{\Gamma}; \quad \mathbf{Q} = (\boldsymbol{I} - \boldsymbol{B})^{-1}\Psi(\boldsymbol{I} - \boldsymbol{B})^{-\mathrm{T}}; \tag{14}$$

$$\boldsymbol{C} = \boldsymbol{\Lambda}; \quad \boldsymbol{D} = \boldsymbol{K}; \quad \boldsymbol{R} = \boldsymbol{\Theta}; \tag{15}$$

The state space matrix is listed on the left of each equation with the structural equation matrix or matrices on the right. Thus, SEMs are SSMs with zero dynamics. If the restriction that $\boldsymbol{A} = \mathbf{0}_{l\times l}$ is relaxed to allow for autoregressive dynamics, then the unified SEM (Kim, Zhu, Chang, Bentler, & Ernst, 2007) is obtained. If the dynamics matrix is further relaxed to allow it to be deterministically time-varying and depend on the covariates (i.e., $\boldsymbol{A} = \boldsymbol{A}_t = \boldsymbol{A}(\boldsymbol{u}_t)$) then the extended unified SEM (Gates, Molenaar, Hillary, & Slobounov, 2011) is obtained. These are important subclasses of SSMs that are starting to be used in the neuroimaging literature. So far they have used SEM software, but their expression and estimation might be simplified if SSM software were used instead.

ON NOTATION

SSMs have been in existence and heavy use since their inception in the 1960s. Because of this history, a number of different notations have developed. Similarly, because of the relation of SSMs to SEMs, some structural equation modeling notation could also be adapted for SSMs.

Table 1 shows six different notations for SSMs. Durbin and Koopman (2001) provided a popular treatment on time series and SSMs and have companion software that implements their methods. West and Harrison (1997) gave a more

Bayesian perspective on dynamic models. Åström and Murray (2010) wrote an introductory book on feedback control systems engineering. Kalman (1960) and Kalman and Bucy (1961) put forth the original papers on SSMs in mechanical and electrical engineering.

Of particular note in Table 1 is the fact that several of the same symbols appear as different conceptual units in different notations. For example, $\boldsymbol{H}_t$ is the factor loadings matrix (measurement model) in Kalman's notation, but it is the observation noise covariance (manifest error covariance) in the Durbin and Koopman model. Similarly, $\boldsymbol{F}_t$ appears in every notation except LISREL and OpenMx, and in each it represents a different part of the model.

Other formulations emphasize different portions of the models. Table 2 shows much of the same information as Table 1, but emphasizes the latent process model, which is sometimes called the state equation. Table 3 compares the measurement models with varying notations.

Notice that the notations for SSMs are numerous and inconsistent. The general form of the model is always the same, but the letters used vary. There is the most agreement between the engineering (Åström & Murray, 2010; Grewal & Andrews, 2008; MATLAB, 2014) notation and the original (Kalman, 1960; Kalman & Bucy, 1961) literature. For this reason, the notation for OpenMx was picked in a way that seemed easiest to remember and most consistent with other notations. It is generally alphabetic: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{Q}, \boldsymbol{R}$. In addition to these matrices, the model also requires the input and covariate variables ($\mathbf{u}_t$) to be specified along with the initial latent mean and variance ($\boldsymbol{x}_0$ and $\boldsymbol{P}_0$).[1] The notational situation of SSMs is in contrast to structural equation modeling, in which different forms of the model are used that turn out to be equivalent (see, e.g., Horn & McArdle, 1980, for the equivalence of the RAM and LISREL notations). The unifying aspect of these SSMs is their treatment of time.

ON TIME

The main benefit of state space modeling is the intrinsic special treatment of time. Modeling dynamics requires some form of treating time differently. SSMs do that, and they can use latent variables. Other forms of modeling dynamics with latent variables (e.g., latent differential equations, dynamic factor analysis, *P*-technique factor analysis, *N*-way factor analysis, latent difference scores, latent growth curves, etc.) all exploit preexisting techniques in various ways to force them to incorporate dynamic or temporal information. The exploitation works, but often has severe limitations. By contrast, SSMs were designed from the ground up to incorporate latent variables and nonstationary dynamics. With state space modeling there is no need to create a chimera to allow for dynamic modeling or measurement error, as these are native to the technique.

State space modeling, as implemented here, uses a type of recursive filter called a Kalman filter. The Kalman filter consists of alternating prediction and correction steps. The prediction step begins from some latent state vector at time $t-1$ called $\boldsymbol{x}_{t-1|t-1}$ along with its error covariance matrix called $\boldsymbol{P}_{t-1|t-1}$, and creates a prediction or forecast for the state vector and covariance at the next time t. Thus, it uses the current estimates $\boldsymbol{x}_{t-1|t-1}$ and $\boldsymbol{P}_{t-1|t-1}$ to create forecasts for the next estimates $\boldsymbol{x}_{t|t-1}$ and $\boldsymbol{P}_{t|t-1}$.

As a set of equations, the predict step is

$$\boldsymbol{x}_{t|t-1} = \boldsymbol{A}\boldsymbol{x}_{t-1|t-1} + \boldsymbol{B}\boldsymbol{u}_t \tag{16}$$

$$\boldsymbol{P}_{t|t-1} = \boldsymbol{A}\boldsymbol{P}_{t-1|t-1}\boldsymbol{A}^T + \boldsymbol{Q} \tag{17}$$

The predict step consists merely of finding the model-predicted mean and covariance matrix for the latent variables at the next time point conditional on their current estimates. Thought of in a probability sense, the predict step maps the current multivariate Gaussian probability distribution to a model-implied prediction of the next probability distribution. The predict step depends only on the latent variable estimates, the model matrices ($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{Q}$), and the measured covariates ($\boldsymbol{u}_t$).

On the other hand, the update step (measurement update, or correct step) depends more on the measured data and its corresponding measurement model. The update step uses the observed data to correct the forecast from the predict step. Thus, it uses the forecast estimates $\boldsymbol{x}_{t|t-1}$ and $\boldsymbol{P}_{t|t-1}$ along with the data at time t to create updated estimates $\boldsymbol{x}_{t|t}$ and $\boldsymbol{P}_{t|t}$.

As a set of equations, the update step is

$$\hat{\boldsymbol{y}}_t = \widehat{\text{Mean}}(\mathbf{y}_t) = \boldsymbol{C}\boldsymbol{x}_{t|t-1} + \boldsymbol{D}\boldsymbol{u}_t \tag{18}$$

$$\tilde{\boldsymbol{y}}_t = \widehat{\text{Residual}}(\boldsymbol{y}_t) = \boldsymbol{y}_t - \hat{\boldsymbol{y}}_t \tag{19}$$

$$\hat{\boldsymbol{S}}_t = \widehat{\text{Cov}}(\mathbf{y}_t) = \boldsymbol{C}\boldsymbol{P}_{t|t-1}\boldsymbol{C}^{\mathrm{T}} + \boldsymbol{R} \tag{20}$$

$$\boldsymbol{K} = \boldsymbol{P}_{t|t-1}\boldsymbol{C}^T\hat{\boldsymbol{S}}_t^{-1} \tag{21}$$

$$\boldsymbol{x}_{t|t} = \boldsymbol{x}_{t|t-1} + \boldsymbol{K}\tilde{\boldsymbol{y}}_t \tag{22}$$

$$\boldsymbol{P}_{t|t} = \boldsymbol{P}_{t|t-1} - \boldsymbol{K}\boldsymbol{C}\boldsymbol{P}_{t|t-1} \tag{23}$$

The update step consists of adjusting the estimates from the predict step based on how well they correspond to the measured data, $\boldsymbol{y}_t$. The matrix $\boldsymbol{K}$, called the Kalman gain, is set so that under certain regularity conditions the updated

[1] In some situations, such as the latent growth curve example, the initial latent mean and variance can be freely estimated from the data. Many times however, the $\boldsymbol{x}_0$ can be estimated but $\boldsymbol{P}_0$ must be fixed. In these cases the typical practice is to fix the initial covariance to a scalar multiple of the identity matrix with the scalar indicating the amount of uncertainty in the initial mean. See, for example, Harvey (1989, pp. 121–122) and Durbin and Koopman (2001, pp. 99–120).

TABLE 1
Notations for Matrices and Vectors in State Space Models Without Inputs

	Durbin & Koopman	LISCOMP	West & Harrison	Åstrom & Murray	Kalman	OpenMx
Observed outputs	$\mathbf{y}_t$	$\mathbf{y}_t$	$\mathbf{a}_t$	$\mathbf{y}_t$	$\mathbf{z}_t$	$\mathbf{y}_t$
Latent states	$\boldsymbol{\alpha}_t$	$\boldsymbol{\eta}_t$	$\boldsymbol{\theta}_t$	$\mathbf{x}_t$	$\mathbf{x}_t$	$\mathbf{x}_t$
Observation noise	$\boldsymbol{\varepsilon}_t$	$\boldsymbol{\varepsilon}_t$	$\mathbf{v}_t$	$\mathbf{w}_t$	$\mathbf{v}_t$	$\mathbf{r}_t$
Dynamic noise	$\mathbf{R}_t\boldsymbol{\eta}_t$	$\boldsymbol{\zeta}_t$	$\mathbf{w}_t$	$\mathbf{v}_t$	$\mathbf{w}_t$	$\mathbf{q}_t$
Observation noise covariance	$\mathbf{H}_t$	$\boldsymbol{\Theta}_t$	$\mathbf{V}_t$	$\mathbf{R}_{wt}$	$\mathbf{R}_t$	$\mathbf{R}_t$
Dynamic noise covariance	$\mathbf{R}_t\mathbf{Q}_t\mathbf{R}_t^T$	$\boldsymbol{\Psi}_t$	$\mathbf{W}_t$	$\mathbf{F}_t\mathbf{R}_{vt}\mathbf{F}_t^T$	$\mathbf{Q}_t$	$\mathbf{Q}_t$
Factor loadings	$\mathbf{Z}_t$	$\boldsymbol{\Lambda}_t$	$\mathbf{F}_t$	$\mathbf{C}_t$	$\mathbf{H}_t$	$\mathbf{C}_t$
Autoregressive dynamics	$\mathbf{T}_t$	$\mathbf{B}_t$	$\mathbf{G}_t$	$\mathbf{A}_t$	$\mathbf{F}_t$, $\boldsymbol{\Phi}_t$	$\mathbf{A}_t$
Prediction error estimate	$\mathbf{v}_t$	$\mathbf{y}_t - \boldsymbol{\Lambda}_t\widehat{\boldsymbol{\eta}}_t$	$\mathbf{a}_t - \mathbf{F}_t\widehat{\boldsymbol{\theta}}_t$	$\mathbf{y}_t - \mathbf{C}_t\widehat{\mathbf{x}}_t$	$\widetilde{\mathbf{z}}_t$	$\widetilde{\mathbf{y}}_t$
Prediction error covariance	$\mathbf{F}_t$	$\mathbf{G}_t$	$\mathbf{C}_t$	-	$\mathbf{S}_t$	$\mathbf{S}_t$

TABLE 2
Latent Process Model Under Different Notations

Durbin & Koopman	$\boldsymbol{\alpha}_t = \mathbf{T}_t\boldsymbol{\alpha}_{t-1} + \mathbf{R}_t\boldsymbol{\eta}_t$
LISCOMP	$\boldsymbol{\eta}_t = \mathbf{B}_t\boldsymbol{\eta}_{t-1} + \boldsymbol{\Gamma}_t\mathbf{x}_t + \boldsymbol{\zeta}_t$
West & Harrison	$\boldsymbol{\theta}_t = \mathbf{G}_t\boldsymbol{\theta}_{t-1} + \mathbf{w}_t$
Kalman	$\mathbf{x}_t = \mathbf{F}_t\mathbf{x}_{t-1} + \mathbf{B}_t\mathbf{u}_t + \mathbf{w}_t$
Åstrom & Murray	$\mathbf{x}_t = \mathbf{A}_t\mathbf{x}_{t-1} + \mathbf{B}_t\mathbf{u}_t + \mathbf{v}_t$
OpenMx	$\mathbf{x}_t = \mathbf{A}_t\mathbf{x}_{t-1} + \mathbf{B}_t\mathbf{u}_t + \mathbf{q}_t$

TABLE 3
Measurement Model Under Different Notations

Durbin & Koopman	$\mathbf{y}_t = \mathbf{Z}_t\boldsymbol{\alpha}_t + \boldsymbol{\varepsilon}_t$
LISCOMP	$\mathbf{y}_t = \boldsymbol{\Lambda}_t\boldsymbol{\eta}_t + \mathbf{K}_t\mathbf{u}_t + \boldsymbol{\varepsilon}_t$
West & Harrison	$\mathbf{a}_t = \mathbf{F}_t\boldsymbol{\theta}_t + \mathbf{v}_t$
Kalman	$\mathbf{z}_t = \mathbf{H}_t\mathbf{x}_t + \mathbf{v}_t$
Åstrom & Murray	$\mathbf{y}_t = \mathbf{C}_t\mathbf{x}_t + \mathbf{D}_t\mathbf{u}_t + \mathbf{w}_t$
OpenMx	$\mathbf{y}_t = \mathbf{C}_t\mathbf{x}_t + \mathbf{D}_t\mathbf{u}_t + \mathbf{r}_t$

latent error covariance matrix $\boldsymbol{P}_{t|t}$ is as small as possible; that is, it has minimum trace. This achieves the goal of having as precise (i.e., smallest variance) an estimate of the latent variables as possible with the given information. The update step is critical for preventing prediction errors from accumulating, especially for nonstationary time series.

The likelihood of an SSM is calculated by prediction error decomposition (PED). PED is merely the multivariate normal likelihood of the data row $\mathbf{y}_t$ given the mean and covariance matrix at row t: Equations 18 and 20, respectively. Thus, the likelihood of an SSM is calculated the same as the likelihood for a SEM:

$$\begin{aligned} -2LL_t = n\log(2\pi) + \log\left|\hat{\boldsymbol{S}}_t\right| \\ + (\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t)^T\hat{\boldsymbol{S}}_t^{-1}(\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t) \end{aligned} \tag{24}$$

where n is the number of observed variables. The primary distinction to be made between the SEM and the SSM likelihoods is that the mean and covariance matrix for SEMs are typically the same for each row of data whereas they are generically different for different rows in SSMs. Critically, because the likelihood equations are essentially identical between SEMs and SSMs, in the OpenMx implementation of SSMs partially missing data are handled modularly by the same full information maximum likelihood (FIML) procedures. Hence, these SSMs are FIML SSMs.

It should be noted, however, that in spite of having a time-varying mean and covariance structure, many SSMs approach a steady state mean and covariance. For stationary latent processes[2] the covariance structure asymptotically approaches a steady state exponentially fast. The asymptotic latent covariance matrix is

$$\operatorname{vec}(\boldsymbol{P}_\infty) = (\boldsymbol{I} - \boldsymbol{A}\otimes\boldsymbol{A})^{-1}\operatorname{vec}(\boldsymbol{Q}) \tag{25}$$

where the vec() operator concatenates all the elements of a matrix into a vector, the $\otimes$ operator is the Kronecker product, and $\boldsymbol{I}$ is the identity matrix. Equation 25 is the solution to the discrete Lyapunov equation from control theory (Kitagawa, 1977). It is the multivariate analog of the asymptotic variance of a univariate lag-one autoregressive process.

$$x_t = a\,x_{t-1} + \varepsilon_t \tag{26}$$

which, under the assumption of stationarity (i.e., $|a|<1$), has variance

$$\sigma_x^2 = \frac{\sigma_\varepsilon^2}{1 - a^2} \tag{27}$$

where σ_x^2 is the variance of the random variable x, a is the autoregressive coefficient for the process, and σ_ε^2 is the residual variance on the lag-one process.

[2] A latent process will have a stationary covariance structure when the (complex) magnitude of the eigenvalues of the dynamics, $\boldsymbol{A}$, are all less than 1.0. The process will have a stationary mean structure when the inputs, u are all zero, and the covariance structure is stationary. Both of these conditions also assume that none of the model matrices are time-dependent (i.e., $\boldsymbol{A}_t = \boldsymbol{A}, \boldsymbol{C}_t = \boldsymbol{C}$, etc.).

When the latent process has a stationary covariance structure, the observed process will also.

$$\hat{\boldsymbol{S}}_\infty = \widehat{\mathrm{Cov}}(\mathbf{y}_\infty) = \boldsymbol{C}\boldsymbol{P}_\infty \boldsymbol{C}^T + \boldsymbol{R} \tag{28}$$

This stationary covariance structure can be useful to conceptualize for people who are accustomed to thinking of static SEMs that have some fixed mean and covariance structure. SSMs, by contrast, have a time-varying mean and covariance structure that nonetheless approaches a fixed structure in the long run for stationary processes. For nonstationary processes, the fixed covariance structure of Equation 25 is not defined. For example, consider the special case of Equation 25 written as Equation 27 when the autoregressive coefficient a is 1.0. This corresponds to an SSM with an eigenvalue on the complex unit circle and is not stationary. When $a = 1$, this requires division by zero in Equation 27; when $|a|>1$, this implies a negative variance. Thus, nonstationary SSMs do not converge to a steady state variance. These are nonstationary SSMs due to the dynamics, $\boldsymbol{A}$, being unstable. Unstable dynamics might still be applicable to empirical phenomena. Exponential growth, for instance, has numerous applications throughout many fields of science, yet is an unstable dynamic process.

Another kind of nonstationary process derives from the process itself varying over time. The dynamics, $\boldsymbol{A}$, might vary as a function of time $\boldsymbol{A}_t$ or even as a function of a time-varying covariate $\boldsymbol{A}(\boldsymbol{u}_t)$. This fluctuation might depend on free parameters, but it cannot depend on other latent states. Dependence on other latent states would make the dynamics stochastically time-varying, which is not possible in the linear SSM; that requires a nonlinear dynamics modeling framework such as dynr (Ou, Hunter, & Chow, 2017). Only deterministically time-varying dynamics are possible in the current OpenMx implementation. We next consider further details of the state space implementation in OpenMx.

ON MODULARITY

There are several other stand-alone programs and packages that can fit SSMs. Stand-alone programs include SsfPack (Koopman, Shephard, & Doornik, 1999) and MKFM6 (Dolan, 2005). The R software also has packages that estimate some SSMs: dlm (Petris, 2010), KFAS (Helske, 2016a; 2016b), dse (Gilbert, 2006), and pomp (King, Nguyen, & Ionides, n.d.), to name just a few. MATLAB (MATLAB, 2014) also has numerous utilities for state space modeling and system identification (Grewal & Andrews, 2001; Hartikainen, Solin, & Särkkä, 2011; MATLAB, 2014). From the standpoint of the structural equation modeler, each of these alternative programs has some drawbacks. Primary among these is the incorporation of multiple individuals. Generally, each of these programs only has the ability to estimate parameters for a single multivariate time series. Moreover, some of the R packages are relatively slow to estimate models compared to OpenMx.[3] Thus, estimation time can be a hurdle for using SSMs. Likewise, learning and perhaps purchasing an entirely new program for state space modeling represents a significant obstacle for many researchers interested in getting started with these kinds of models. Hence, there are certainly benefits to a fast-running state space modeling program in a free, open-source environment in which SEMs can be specified with roughly the same syntax as SSMs. The OpenMx implementation of SSMs satisfies these requirements.

A major benefit of the OpenMx implementation of SSMs is their modularity. Because this SSM implementation is embedded in a high-level structural equation modeling environment, many features are available for SSMs that might be difficult to implement in a stand-alone program or package. In the implementation FIML is used for missing data handling. The same underlying code is used for both SEMs and SSMs. Likewise, the same maximum likelihood fit function (`mxFitFunctionML`) is actually used. The SEM and SSMs routines simply have different ways of computing the expected means vector and covariance matrix for each row of data. Once these are obtained, the actual likelihood functions are the same, so the same code is reused.

For the user, model specification of SSMs follows the same syntax as SEMs in OpenMx. Matrices form the backbone of this specification. A matrix is created for OpenMx with the `mxMatrix()` function. Starting or fixed values are entered into the `values` argument; parameter labels are given in the `labels` argument; and each value is indicated as free or not by the the logical `free` argument. Upper and lower bounds on each parameter can also be set with the `ubound` and `lbound` arguments, respectively. For convenience, several prebuilt matrix constructors are available for diagonal, symmetric, zero, unit, and standardized matrices using the type argument with values "Diag," "Symm," "Zero," "Unit," and "Stand," respectively.

The matrix specification is extended by the inclusion of arbitrary algebraic expressions OpenMx calls "algebras" with the `mxAlgebra` function. Algebras can involve expressions of matrices, free parameters, and other algebras. The possible expressions encompass a wide variety of mathematical functions including but not limited to addition, matrix multiplication, scalar multiplication, matrix inverse,

[3] In several tests using the same model and data, dlm was 10 to 25 times slower than OpenMx. For a full example that fits the same model to the same data using OpenMx and dlm, see https://github.com/OpenMx/OpenMx/blob/master/inst/models/passing/StateSpaceOsc.R, in which the same parameter estimates are obtained but OpenMx is about 20 times faster (1.5 sec vs. 30 sec on a Windows 10 64-bit machine with an Intel i7-6500U processor). Care must be taken here to compare the OpenMx version of the model that does not estimate the standard errors or Hessian because the optim routine that dlm uses does not do this.

extracting eigenvalues, Cholesky decompositions, trigonometric functions, the Kronecker product, matrix exponentiation, and multivariate normal integration. Algebras allow highly complex models to be estimated in OpenMx. By placing an SSM implementation within OpenMx, all of this functionality is also available for SSMs.

Simple parameter equality constraints are achieved by giving two or more free parameters the same labels value. Nonlinear equality constraints can almost always be accomplished with the `mxAlgebra` by letting the quantities of interest be the result of an algebra. These can also be accomplished with the `mxConstraint` function at the cost of additional optimization complexity and the loss of standard errors.[4] Nonlinear inequality constraints (e.g., keeping the eigenvalues of the dynamics matrix within the unit circle) can rely on the `mxConstraint` feature. Finally, definition variables allow elements from the data to directly enter into and modify the model in ways only limited by the modeler's imagination.

Model summary for SSMs has many of the same benefits of modularity as model specification. Without the user having to learn new functions (or the developer write them), model summary is much the same for SSMs and SEMs in OpenMx. First the summary method that is defined for all models in OpenMx works identically for SSMs. Relative fit indexes such as Akiake's information criterion (AIC) and Bayesian information criterion (BIC) are defined in the same way across SSMs and SEMs. Absolute fit indexes like root mean square error of approximation (RMSEA), chi-squared, comparative fit index (CFI), and Tucker–Lewis Index (TLI) could also be defined conditional on the users having a specified "saturated" and "independence" model to serve as the basis of comparison. Standard errors based on the observed Fisher information (i.e., the Hessian of the negative log-likelihood function) are included and use identical underlying code as the standard errors for SEMs. Moreover, a valuable feature of OpenMx is the availability of profile-likelihood confidence intervals (Neale & Miller, 1997; Pritikin, Rappaport, & Neale, 2017). Profile-likelihood confidence intervals tend to provide better inference for bounded parameters, especially when near their bounds (Pek & Wu, 2015; Wu & Neale, 2012). This can be critically important when trying to establish whether or not a variance parameter is zero. The `mxCompare` function that performs likelihood ratio testing is also available for testing general nested models as both SEMs and SSMs.

[4] Standard errors are not available for models that use mxConstraint because the second derivative matrix (Hessian) of the likelihood function does not respect constraints, thus violating assumptions of the likelihood theory that defines the large sample variance of the parameters (e.g., Fisher, 1925).

Two recently added features of OpenMx further enhance user capabilities in OpenMx relative to other SSM packages or programs. First, modification indexes (Sörbom, 1989) are defined with the `mxMI` function for any model that uses maximum likelihood, thus including SSMs. Second, the `mxSE` function allows the user to compute the standard error on any algebraic expression using the well-known delta method. This could be useful when the sum of two parameters is of interest in addition to their individual values. To our knowledge no other existing program that allows SSMs has modification indexes or allows automatic computation of standard errors for arbitrary algebraic expression, especially multivariate expressions.

As outlined earlier, the modular implementation of SSMs in an SEM environment allows numerous features to be available and identical across both modeling frameworks. Once a user has learned about SEM in OpenMx, it is an easy step to extend this to state space modeling. The model specification for matrix-type models is identical. The same kind of matrices, algebras, and constraints are possible in both frameworks. Model summary and inference have many of the same capacities for relative fit indexes, standard errors, profile-likelihood confidence intervals, and model comparison. Recently added features of OpenMx apply equally well to SSMs and SEMs. This lets modification indexes work for SSMs which, to our knowledge, has never before been done. It also permits standard errors for any function of parameters expressible in OpenMx's very general algebras. The wide array of features available for SSMs in OpenMx greatly extend the utility of SSMs for advanced SEM.

EXAMPLES

Given the long history and greatly flexibility of state space modeling, many applications are possible. From lunar navigation (Smith, Schmidt, & McGee, 1962) to electromyography (Yang & Chow, 2010), the sky is the limit for state space modeling. With so many potential areas of application, we instead focus on several representative examples of SSMs, discussing the meaning of each example in turn and illustrating the features of the OpenMx implementation along the way. We begin with a lag-one latent autoregression (i.e., the direct autoregressive factor score model of Nesselroade, McArdle, Aggen, & Meyers, 2002). We then progress to multiple lag models, and multiple lag moving average models. We conclude with two models for multiple subjects: one in an autoregressive form and another in the form of a latent growth curve.

Lag-One Latent Autoregression

A simple first example of an SSM is a lag-one autoregression. Often, this kind of model is called dynamic factor analysis. Such a model might be produced by a system as follows:

$$\boldsymbol{\eta}_t = a\ \boldsymbol{\eta}_{t-1} + q_t \qquad q_t \sim \mathcal{N}(0,1) \tag{29}$$

where $\boldsymbol{\eta}_t$ is the latent variable at time t, a is the autoregressive coefficient, and q_t is the normally distributed dynamic noise with zero mean (by definition) and variance 1 (by choice). Equation 29 is a typical lag-one autoregression. Because $\boldsymbol{\eta}_t$ is a latent variable, we also provide a measurement model for it, just as if we were specifying an SEM.

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix}_t = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \end{pmatrix} \boldsymbol{\eta}_t + r_t$$
$$r_t \sim \mathcal{N}\left(0, \begin{pmatrix} \sigma^2_{r_1} & & & & \\ & \sigma^2_{r_2} & & & \\ & & \sigma^2_{r_3} & & \\ & & & \sigma^2_{r_4} & \\ & & & & \sigma^2_{r_5} \end{pmatrix}\right) \tag{30}$$

Equation 30 is a standard measurement model for a one-factor model. All factor loadings are freely estimated, so the variance of q_t in Equation 29 was fixed to 1. In many cases, the dynamic noise variance can be estimated but it is fixed here for an easier analogy to factor analysis. If the a parameter in Equation 29 were fixed to zero, a typical one-factor model would result and the dynamic noise, $\text{Var}(q_t)$, would be equal to the factor variance, $\text{Var}(\boldsymbol{\eta}_t)$.

The SSM given by Equations 29 and 30 can be estimated with the following R code.

```
require(OpenMx)
data(demoOneFactor)
nvar <- ncol(demoOneFactor)
unam <- colnames(demoOneFactor)
```

First, the OpenMx package and the data set called demoOneFactor included as part of the OpenMx package are loaded. Next, the matrices for the SSM are specified using the OpenMx matrix syntax. The ***A*** matrix that gives the autoregressive dynamics is specified with the names of the arguments included, but subsequent matrices exclude these argument names for brevity.

```
amat <- mxMatrix(type = "Full", nrow = 1,
  ncol = 1, free = TRUE,
values = .3, name = "A", labels = "a")
bmat <- mxMatrix("Zero", 1, 1, name = "B")
clab <- paste0("load", 1:nvar)
cdim <- list(unam, "F1")
cmat <- mxMatrix("Full", nvar, 1, TRUE, .6,
  name = "C",
dimnames = cdim, labels = clab)
dmat   <-   mxMatrix("Zero",   nvar,   1,
  name = "D")
qmat <- mxMatrix("Diag", 1, 1, FALSE, 1,
  name = "Q")
rlab <- paste0("resid", 1:nvar)
rmat <- mxMatrix("Diag", nvar, nvar, TRUE,
  .2, name = "R", labels = rlab)
xmat <- mxMatrix("Zero", 1, 1, name = "x0")
pmat <- mxMatrix("Diag", 1, 1, FALSE, 1,
  name = "P0")
umat <- mxMatrix("Zero", 1, 1, name = "u")
```

With all the SSM matrices created, we next include these matrices in a model, indicated as an SSM by the use of the mxExpectationStateSpace. The model will use maximum likelihood estimation (mxFitFunctionML).

```
ssModel <- mxModel(model = "Lag1LAR",
   amat, bmat, cmat, dmat, qmat, rmat, xmat,
     pmat, umat,
   mxData(observed     =     demoOneFactor,
     type = "raw"),
   mxExpectationStateSpace("A", "B", "C",
     "D", "Q", "R", "x0", "P0", "u"),
   mxFitFunctionML()
)
ssRun <- mxRun(ssModel)
summary(ssRun)
```

The last line is a call to the summary generic function. It is the same function used to obtain summary information for linear regression, mixed effects models, and even data sets. The summary method for MxModel objects produces output of the following form.

```
Summary of lag1LAR

free parameters:

    Name    matrix  row  col  Estimate    Std.Error A
1   a11     A       1    1    0.07532402  0.045519534
2   load1   C       x1   F1   0.39760087  0.015530191
3   load2   C       x2   F1   0.50383630  0.018202434
4   load3   C       x3   F1   0.57771453  0.020428969
5   load4   C       x4   F1   0.70211309  0.023974543
6   load5   C       x5   F1   0.79680809  0.026647966
7   resid1  R       1    1    0.04076104  0.002806575
8   resid2  R       2    2    0.03790698  0.002795323
9   resid3  R       3    3    0.04074343  0.003144651
10  resid4  R       4    4    0.03953963  0.003408718
11  resid5  R       5    5    0.03612797  0.003668308
```

```
Model Statistics:

                 | Parameters    | Degrees of    | Fit (-2lnL
                                     Freedom         units)
Model:                11            2489          936.7202
Saturated:            20            2480             NA
Independence:         10            2490             NA
Number of observations/statistics: 500/2500

Information Criteria:

         | df Penalty      | Parameters       | Sample-Size
                               Penalty           Adjusted
AIC:       -4041.28         958.7202              NA
BIC:      -14531.44        1005.0809           970.1662

To get additional fit indices, see help
  (mxRefModels)
timestamp: 2017-08-09 13:22:50
Wall clock time (HH:MM:SS.hh): 00:00:03.07
optimizer: CSOLNP
OpenMx version number: 2.7.11
Need help? See help(mxSummary)
```

The summary is spaced into separate sections on the parameters, model statistics, information criteria, and further details like the running time and version of OpenMx used. Importantly, the general form of the model summary is the same for SEMs and SSMs.

Table 4 compares the parameter estimates obtained from OpenMx to those produced by MKFM6 (Dolan, 2005). With the five decimals reported for parameter estimates by MKFM6, there is one minor difference between the two: The factor loading for the third indicator is off by 0.00001. Furthermore, the standard errors match down to the four decimals reported by MKFM6.

From the summary and inspection of Table 4 it is apparent that the latent autoregressive coefficient is near zero. This should be the case because the data used are generated from a one-factor model whose rows should be independent, and thus have a zero dynamics. The user might want to test the hypothesis that the dynamics are zero. This can be done with nested model comparisons.

```
ssZero         <-         mxModel(ssModel,
  name = 'Lag0LAR',
mxMatrix("Zero", 1, 1, name = "A")
)
ssZeroRun <- mxRun(ssZero)
mxCompare(ssRun, ssZeroRun)
# producing output
```

```
     base       comparison    ep     minus2LL      df      AIC        diffLL      diffdf      p
1    Lag1LAR    <NA>          11     936.7202     2489    -4041.28    NA          NA          NA
2    Lag1LAR    Lag0LAR       10     939.4503     2490    -4040.55    2.73008     1           0.09847433
```

TABLE 4
Comparison of OpenMx and MKFM6 Estimates and Standard Errors (*SE*) for a Lag-One Latent Autoregressive Model

		OpenMx		*MKFM6*	
Parameter	*Matrix*	*Estimate*	*SE*	*Estimate*	*SE*
a	**A**	0.07532	0.04552	0.07532	0.0455
λ_1	**C**	0.39760	0.01553	0.39760	0.0155
λ_2	**C**	0.50384	0.01820	0.50384	0.0182
λ_3	**C**	0.57771	0.02043	0.57772	0.0204
λ_4	**C**	0.70211	0.02397	0.70211	0.0240
λ_5	**C**	0.79681	0.02665	0.79681	0.0266
$\sigma^2_{r_1}$	**R**	0.04076	0.00281	0.04076	0.0028
$\sigma^2_{r_2}$	**R**	0.03791	0.00280	0.03791	0.0028
$\sigma^2_{r_3}$	**R**	0.04074	0.00314	0.04074	0.0031
$\sigma^2_{r_4}$	**R**	0.03954	0.00341	0.03954	0.0034
$\sigma^2_{r_5}$	**R**	0.03613	0.00367	0.03613	0.0037

which yields $\chi^2(df = 1) = 2.73$, $p = 0.098$. The same hypothesis could be tested with profile likelihood-based confidence intervals.

```
ssCI <- mxModel(ssModel,
mxCI("A")
)
ssCIRun <- mxRun(ssCI, intervals = TRUE)
summary(ssCIRun)
```

which adds the following new section to the summary.

```
confidence intervals:
        lbound        estimate     ubound note
a11  -0.01414349   0.07532402   0.1647398
```

and gives the same point estimate as before (**A** = 0.075), but now also provides confidence limits based on the change in the likelihood function: 95% CI = (−0.014, 0.165). Note that an SSM with **A** as a zero matrix is identical to a factor analysis.

Kalman Scores

One of the benefits of the Kalman filter for fitting dynamic linear models is the calculation of Kalman scores. Kalman scores are estimates of the latent variable scores: factor scores. In fact, when the dynamics matrix, A, is zero—thus creating a factor analysis—the Kalman scores are identical to regression-based factor scores (Priestley & Subba Rao, 1975). For nonzero dynamics, the Kalman filter produces two latent variable scores: One is the predicted score ($\boldsymbol{x}_{t|t-1}$, Equation 16), and the other is the updated score ($\boldsymbol{x}_{t|t}$, Equation 22). Beyond the mere score estimation, the Kalman filter also produces an error covariance of the latent score: these are $\boldsymbol{P}_{t|t-1}$ (Equation 17) for the error covariance of the prediction score and $\boldsymbol{P}_{t|t}$ (Equation 23) for the error covariance of the correction score. These variance matrices provide a sense of the precision and reliability of the latent variable score estimates. However, the Kalman scores are based solely on the information available up to time $t-1$ or time t. There also exist latent variable score estimates based on the whole time series. These are the so-called Rauch, Tung, Striebel (RTS; Rauch, Tung, & Striebel, 1965) smoothed scores and covariances. All three types of scores and covariances are available in OpenMx by using the `mxKalmanScores` function. An example follows that uses the same model from the initial example.

```
scores <- mxKalmanScores(ssRun)
```

Data Generation From a Model

Through the use of generic functions, data from many models can be simulated in OpenMx. The generic data generation functions have a method built for SSMs. Thus, data can be generated from any SSM specified in OpenMx.

```
fakeData <- mxGenerateData(ssRun,
  nrows = 200)
# creates a numeric matrix with 200 rows and
  5 columns
# according to the model coefficients in
  ssRun
head(fakeData)
# produces output

         x1          x2         x3          x4         x5
1  0.123272030  0.21644810 -0.2012850 -0.09387530 -0.2409484
2  0.001873846 -0.42104294 -0.1414802  0.02554446 -0.1671043
4  0.213666453  0.22900478  0.6447381  0.34408800  0.9456453
5  0.925392627  1.11024982  1.4850415  1.25854234  1.9391823
6 -0.154339012 -0.19926690  0.3784332  0.17625686  0.2715762
```

Of course, the actual values of the simulated data will varying from one call of `mxGenerateData` to the next, but repeatable data generation can be achieved by setting the R pseudorandom number generator seed with the set.seed function. Such data generation based on a model can be extremely useful when conducting simulation studies.

Standard Errors on Arbitrary Expressions

It is often the case that researchers are interested not only in the parameters of their model, but also in functions of these parameters. For example, someone might estimate the means of two variables and also be interested in the difference between these two means. More complicated functions of parameters might likewise have importance to specific hypotheses. The `mxCI` function can compute profile-likelihood confidence intervals on free parameters and on arbitrary expressions defined by the user. One drawback of these confidence intervals is their computational intensity and consequent estimation time. A fast workaround as an approximation to full confidence intervals is the mxSE function, which computes standard errors of free parameters and arbitrary user-defined expressions. From these standard errors, Wald-type confidence intervals (with all their limitations) can be constructed.

As an example of `mxSE` usage, consider a researcher who wanted to know (a) the asymptotic covariance of the latent variables (Equation 25) in the dynamic factor analysis example, and (b) also the standard errors around this covariance. These questions are answered with the following R code.

```
mxEval(solve(1 - A %x% A) %*% cvectorize(Q),
  ssRun)
          [,1]
[1,] 1.005706
mxSE(solve(1 - A %x% A) %*% cvectorize(Q),
  ssRun)
          [,1]
[1,] 0.00693591
```

In this example, the asymptotic latent variance is a 1×1 matrix: $\mathbf{P}_{\infty} = 1.0057$ with 0.0069 standard error. Another possible question unique to state space modeling might be the sign and magnitude of the eigenvalues of the dynamics matrix, which are related to the stability of the dynamics. If the eigenvalues of the discrete-time dynamics are within the unit circle, then the dynamics are stable. The subsequent R code computes the complex magnitude of the eigenvalues for the dynamics.

```
mxEval(sqrt(eigenval(A)%^%2 + ieigenval
  (A)%^%2), ssRun)
mxSE(sqrt(eigenval(A)%^%2 + ieigenval(A)%
  ^%2), ssRrun)
```

Of course, in the one-dimensional case the eigenvalue of A is $A = 0.075$ as estimated previously, and the standard error of the eigenvalue is just the standard error of A: 0.045 as reported in the summary of the model. However, the same technique is readily applicable in higher dimensional problems where the eigenvalues are not so easily obtained.

As a second example of `mxSE` usage, consider the case of parameter transformations. In many situations researchers find value in optimizing one set of parameters but interpreting a different set of parameters. Call the former the working parameters and the latter the natural parameters. A frequent instance of this occurs when variances have a lower bound of zero: A working parameter can be used that is unbounded such that the natural parameter defined as the exponential of the working parameter is bounded. The researcher does not want to interpret the logarithm of the variance or this working standard error, but rather the variance and the standard error of the variance. With minimal code, the dynamic factor analysis example can be modified to estimate a transformed set of residual variances.

```
rlab2 <- paste("logResid", 1:nvar, sep = "")
rmat2 <- mxMatrix("Full", nvar, 1, TRUE, .2,
  name = "logr", labels = rlab2)
ralg  <-  mxAlgebra(vec2diag(exp(logr)),
  name = 'R')

# Remove the old residuals matrix
ssModel2 <- mxModel(model = ssModel,
  name = "TransformedLagOneLatentAutoregr-
        ession",
  "R", remove = TRUE
)

# Add the new residuals
ssModel2 <- mxModel(model = ssModel2,
  rmat2, ralg
)
ssRun2 <- mxRun(ssModel2)
```

The parameter estimates for the residuals that result from this model are log-scaled working parameters. Table 5 shows the working and natural parameters for this model along with their standard errors. The standard errors of the natural parameters can be obtained with

```
mxSE(diag2vec(R), ssRun2)
```

For comparison Table 5 also shows the residual variances and standard errors estimated in the original dynamic factor model that did not use parameter transformations. Note that both the transformed estimates and the transformed standard errors (i.e., natural parameters) match the original estimates

TABLE 5
Using mxSE to Transform Natural and Working Parameter Standard Errors (*SE*)

	Working		*Natural*		*Original*	
Parameter	*Estimate*	*SE*	*Estimate*	*SE*	*Estimate*	*SE*
$\sigma^2_{r_1}$	−3.20003	0.06885	0.04076	0.00281	0.04076	0.00281
$\sigma^2_{r_2}$	−3.27262	0.07374	0.03791	0.00280	0.03791	0.00280
$\sigma^2_{r_3}$	−3.20046	0.07718	0.04074	0.00314	0.04074	0.00314
$\sigma^2_{r_4}$	−3.23045	0.08621	0.03954	0.00341	0.03954	0.00341
$\sigma^2_{r_5}$	−3.32069	0.10154	0.03613	0.00367	0.03613	0.00367

and standard errors down to at least five decimals. It should be noted that OpenMx allows lower and upper bounds on parameters, so this particular transformation was not strictly needed to enforce bounds. However, the two examples of `mxSE` illustrate cases where the objects of research interest are multivariate or univariate transformations of the optimized free parameters and standard errors for these objects can still be obtained.

Before considering further examples of SSMs, a limitation of `mxSE` is worth discussing. The mxSE function depends on the prior existence of standard errors for the original free parameters. Thus, when a model uses `mxConstraint` standard errors are not computed and mxSE cannot be used directly. The function can still be used if the user provides a covariance matrix for the free parameters in the cov argument of the `mxSE` function. As noted previously, `mxConstraint` can usually be replaced with `mxAlgebra`, which accomplishes the same task without impairing the standard errors.

Multiple Lags

In addition to studying how a latent variable at one time point might influence the next time point, latent variables might have influence over the next two, three, or more occasions. The longer range influences are higher order lag effects of the latent variable. Higher order lags can be incorporated into the lag-one structure of SSMs by extending the latent state.[5] The structural model for a lag-four (i.e., AR(4)) model is given in Equation 31.

$$\underbrace{\begin{pmatrix} \xi_t \\ \xi_{t-1} \\ \xi_{t-2} \\ \xi_{t-3} \end{pmatrix}}_{\mathbf{x}_t} = \underbrace{\begin{pmatrix} \phi_1 & \phi_2 & \phi_3 & \phi_4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} \xi_{t-1} \\ \xi_{t-2} \\ \xi_{t-3} \\ \xi_{t-4} \end{pmatrix}}_{\mathbf{x}_{t-1}} + \underbrace{\begin{pmatrix} \varepsilon_t \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{\mathbf{q}_t} \tag{31}$$

From Equation 31 it can be readily seen that $\xi_t = \phi_1\xi_{t-1} + \phi_2\xi_{t-2} + \phi_3\xi_{t-3} + \phi_4\xi_{t-4} + \varepsilon_t$, and thus follows an AR(4) structure. The other parts of the structural model are only present to define the other latent variables as lags of the first latent variable. The ϕ_1 coefficient gives the usual lag-one effect of ξ_{t-1} on ξ_t. Now we have an additional coefficient ϕ_2 for the effect of ξ_{t-2} on ξ_t, and so on up to lag four. The measurement model for the single latent variable ξ_t can take any form. An example is shown here:

[5] Just as higher order differential equations can be represented as a system of first-order differential equations, higher order lags can be represented as a system of first-order lags.

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}}_{y_t} = \underbrace{\begin{pmatrix} l_1 & 0 & 0 & 0 \\ l_2 & 0 & 0 & 0 \\ l_3 & 0 & 0 & 0 \end{pmatrix}}_{C} \underbrace{\begin{pmatrix} \xi_t \\ \xi_{t-1} \\ \xi_{t-2} \\ \xi_{t-3} \end{pmatrix}}_{x_t} + \underbrace{\begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix}}_{r_t} \qquad (32)$$

It is important to note that the measurement of a multiple lag latent variable would usually only be done on the first latent variable, and not the lagged versions of that latent variable. If two latent variables were desired with multiple lags then each of the ϕ_i in Equation 31 would become a 2 × 2 block describing the the lag i vector autoregressions. Similarly, the ones in the subdiagonal would become 2 × 2 identity matrices, the variance of **e** would be 2 × 2, and the λ_i of Equation 32 would become 1 × 2 blocks of factor loadings. A full code example of this AR(4) model is included as supplemental material.

Multiple Lag Moving Average

The multiple lag model can be further extended to include so-called moving averages, or carry-forward effects of the residuals on the latent variable.

$$\underbrace{\begin{pmatrix} \xi_t \\ \xi_{t-1} \\ \xi_{t-2} \\ \xi_{t-3} \\ \eta_t \end{pmatrix}}_{\mathbf{x}_t} = \underbrace{\begin{pmatrix} \phi_1 & \phi_2 & \phi_3 & \phi_4 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & \theta_1 & \theta_2 & \theta_3 & 0 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} \xi_{t-1} \\ \xi_{t-2} \\ \xi_{t-3} \\ \xi_{t-4} \\ \eta_{t-1} \end{pmatrix}}_{\mathbf{x}_{t-1}} + \underbrace{\begin{pmatrix} \varepsilon_t \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{\mathbf{q}_t} \qquad (33)$$

As in the previous example, ξ_t follows an AR(4) pattern, but now a new latent variable η_t is defined. Hamilton (1994, p. 3044) showed that defining the equations like this results in an η_t that follows an ARMA(4, 3) pattern. Of course this is readily extended to the more general ARMA(p, p − 1). Moreover, arbitrary ARMA(p, q) patterns are obtained by creating the larger ARMA(m, m − 1) structure where $m = max(p, q + 1)$ and fixing to zero higher order autoregressions or moving averages as necessary.

The measurement model is parallel to Equation 32 but now the last latent variable is measured instead of the first.

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}}_{y_t} = \underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 & \lambda_1 \\ 0 & 0 & 0 & 0 & \lambda_2 \\ 0 & 0 & 0 & 0 & \lambda_3 \end{pmatrix}}_{\mathbf{C}} \underbrace{\begin{pmatrix} \xi_t \\ \xi_{t-1} \\ \xi_{t-2} \\ \xi_{t-3} \\ \eta_t \end{pmatrix}}_{\mathbf{x}_t} + \underbrace{\begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix}}_{\mathbf{r}_t} \qquad (34)$$

Although these are univariate ARMA models, it could be extended to vector form, VARMA, with block matrices just as with the previous example. The ϕ_i and θ_i become matrices, and instead of a column in **C** there is a multiple factor block in **C**. A final note on the AR(4) and AR(4,3) examples is worth mention. The observed dimension is 3, but the latent dimension is 5; that is, there are more latent variables than manifest. Because of the structure in the dynamics **A** and the measurement **C**, this is entirely possible in SSMs. A full ARMA example is also included as supplemental material in Appendix A.

Multiple Subjects Autoregression

Across a wide variety of settings researchers encounter multiple observations on multiple units. Perhaps the most typical of these is multiple subject repeated measures data. The state space methods in OpenMx can handle this situation, too. The key shift is to consider each person as a group, and each row of data as a time at which the person was measured. In this way, multisubject SSMs are actually multigroup models. We provide here a multisubject latent autoregression as an extension of the first example using the same model.

First, we generate data on 80 subjects with only five observations per person, using the starting values from the first example model.

```
# Generate fake data from the original model
nSubj <- 80
nTime <- 5
dataL <- list()
for(k    in    1:nSubj){dataL[[k]]     <-
  mxGenerateData(ssModel, nTime)}
indivmodels <- list()
modNames <- paste0("indiv", 1:nSubj)
```

Next, we create a multigroup model by looping through all the subjects using the same model for each subject and only changing the data.

```
# Create multisubject model
for(k in 1:nSubj){
      DataSetForSubjectK <- dataL[[k]]
      indivmodels[[k]]     <-     mxModel
        (name = modNames[k],
          model = ssModel,
          mxData(DataSetForSubjectK,
            type = 'raw'))
}
multiSubjModel          <-          mxModel
  (name = "MultiMod", indivmodels,
      mxFitFunctionMultigroup(modNames))
```

Finally, we run the model and examine the results.

```
# Run model and examine results
multiSubjRun <- mxRun(multiSubjModel)
summary(multiSubjRun)
```

```
# True generating paramters
coef(ssModel)

# Estimated paramters from multisubject
  model with
# 80 individuals and five time points each
coef(multiSubjRun)
```

In this example, we estimated a dynamic factor analysis model on 80 people with only five observations per person. Of course, with only five occasions, the model could have been written as a longitudinal factor analysis (e.g., Olsson & Bergman, 1977). However, with the SSM specification it is quite easy to extend the number of people and the number of time points to any arbitrary value without loss of computational efficiency. The multigroup SSM scales linearly in the number of time points and the number of people (Gu et al., 2014), whereas the rate-limiting step in a longitudinal SEM (i.e., the inverse of the expected covariance matrix) scales cubically with the number of time points as was mentioned in the introduction.

Multiple Subjects Latent Growth

As a final example of the SSM, consider the latent growth curve model (McArdle & Epstein, 1987). This model is generally written either as a SEM or as a linear mixed effects model, but it has an SSM form as well. The state equation for the latent growth model has two latent variables, an intercept and a slope. These latent variables have no variability across time, being mapped by an identity matrix from one time to the next.

$$\underbrace{\begin{pmatrix} I_t \\ S_t \end{pmatrix}}_{\mathbf{x}_t} = \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} I_{t-1} \\ S_{t-1} \end{pmatrix}}_{\mathbf{x}_{t-1}} + \underbrace{\begin{pmatrix} 0 \\ 0 \end{pmatrix}}_{\mathbf{q}_t} \tag{35}$$

However, the latent variables have a nontrivial mean and covariance structure at the initial time point.

$$\mathbf{x}_0 \sim \mathcal{N}\left(\begin{pmatrix} \bar{I} \\ \bar{S} \end{pmatrix}, \begin{pmatrix} \sigma_I^2 & \sigma_{IS} \\ \sigma_{IS} & \sigma_S^2 \end{pmatrix} \right) \tag{36}$$

In this case, a multiple group model is estimated in which each person is a group, all free parameters are the same across groups, and each person has a time series of observations. The initial mean and variance for this model represent the variability between persons in the intercept and slope values. The within-person variability only occurs in the measurement part of the model. The measurement model takes a special form that has the time of observation entering into the **C** matrix as a factor loading.

$$\underbrace{(y)}_{\mathbf{y}_t} = \underbrace{(1 \quad t)}_{\mathbf{C}} \underbrace{\begin{pmatrix} I_t \\ S_t \end{pmatrix}}_{\mathbf{x}_t} + \underbrace{(r)}_{\mathbf{r}_t} \tag{37}$$

This creates an identical model to the SEM version of the latent growth curve model, resulting in the same likelihood function and the same final parameter estimates. In fact, there is a general mapping between all two-level linear mixed effects models and SSMs (Jones, 1993) that can also be extended to two-level random intercept SEMs in a computationally efficient way (Gu et al., 2014). Importantly, the random effects estimates from mixed models are equal to the Kalman smoothed scores for this and other state space versions of random effects models. A full script for the SSM latent growth curve model is included as supplemental material in Appendix B. Point estimates for the same model specified as an SSM, an SEM, or a mixed effects model and fit to the same data are also provided in the following R output.

```
             SSM         SEM        mixed
resid    2.3161816   2.3161813   2.3161816
meanI    9.9303038   9.9303036   9.9303036
meanS    1.8133098   1.8133097   1.8133098
varI     3.8786637   3.8786646   3.8786650
covIS    0.4602485   0.4602481   0.4602481
varS     0.2577103   0.2577103   0.2577102
```

The point estimates match down to five or six decimals for all the parameters. The standard errors (not shown) similarly correspond. Note that the mixed effects version of this model was fit with maximum likelihood rather than restricted maximum likelihood so that the variance estimates were evaluated by the same criteria across all three models.

DISCUSSION

The modular implementation of SSMs in the OpenMx SEM environment allows the novice state space modeler to capitalize on their knowledge of SEM and OpenMx when attempting to run dedicated longitudinal models. Modularity also allows the developer to have ready access to a multitude of high-level features with a minimum of new effort to obtain them. This article has introduced the SSM in the context of structural equation modeling. A novel relationship between SSMs and SEMs was shown: Every SEM can be written as an SSM with zero dynamics (refer to Equations 12–15 for full details). Next some computational aspects of the SSM were discussed, and followed by many details of the modular SSM implementation in OpenMx. Finally, several examples showed code that specifies typical kinds of SSMs: one and multiple lag autoregression, moving averages, multiple subjects models, and latent growth models.

Perhaps the largest shortcoming of the current SSM implementation is the exclusion of ordinal or categorical observed variables. There are methods for ordinal variables in OpenMx (e.g., Neale et al., 2016; Pritikin, Hunter, & Boker, 2015) but they have not yet been combined with SSMs in OpenMx. Such a combination would extend the SSM for Gaussian observed variables as described here to an SSM for exponential family variables as described by Fahrmeir (1992) and Fahrmeir and Wagenpfeil (1997), which are implemented in the R package KFAS (Helske, 2016a). These extensions require a generalization of the likelihood function (Equation 24) to account for the non-Gaussian nature of the observations coupled with their nonindependence as structured in the state equation (Equation 8).

Although examples of random effects models and multisubject models were provided, the current implementation requires a model for each person. In R it is relatively easy to programmatically create hundreds of models with a simple for loop; still, even advanced modelers might desire a cleaner interface for modeling multiple people. Likewise, a connection to the built-in multilevel modeling in OpenMx (Pritikin, Hunter, Von Oertzen, Brick, & Boker, 2017) would make the random effects much easier to create and more general across many levels.

A helpful feature of SEMs that could be extended to SSMs is the use of paths diagrams. Diagrams for SEMs quickly communicate a large amount of information about a model, and if available, could increase the adoption of SSMs for intensive longitudinal data analysis. We have not yet seen any proposed diagramming method for SSMs, but it seems that one could be created. For the measurement part of SSMs, the identical diagramming rules could be carried over from SEMs. The path tracing rules would not need any modification to depict measurement for SSMs. The structural part of the SSM, on the other hand, would need some other kind of path that moves across time. The across-time covariance structure for the latent variables follows the path tracing rules, but there would be a need to combine the across-time covariance structure and the within-time covariance structure according to new rules that have not yet been developed.

A final limitation of this article is the omission of continuous-time modeling (Driver, Oud, & Voelkle, 2015; Oud, 2002; Singer, 2011; Voelkle, Oud, Von Oertzen, & Lindenberger, 2012). In point of fact, the OpenMx implementation of SSMs includes continuous-time SSMs. A full description of the continuous-time SSM is beyond the scope of this article, but all the same OpenMx functionality and features are available for continuous-time models as were described here for discrete-time models. The model specification is almost identical. The user need only add the t argument to the `mxExpectationStateSpace`, which gives a 1×1 matrix that links to a column in the data via definition variables. The data column gives the times at which measurement occurred. This small change in the code shifts from discrete-time modeling to continuous-time modeling. No additional changes are required. Caution is warranted when making this switch, however, because although the same model specification is possible for both continuous- and discrete-time models, the interpretation of model matrices and parameters for the continuous-time state equation is quite different from that for the discrete-time version.

The primary aim of this article was to introduce state space modeling to people who are already quite familiar with structural equation modeling. Having both SSMs and SEMs available in the same program with the same specification language greatly eases this introduction. With several example scripts included in this article, we hope that users who are interested in state space modeling will run the examples, feel comfortable extending them, and subsequently apply them to their own work in the many novel ways afforded by the free and open-source implementation.

REFERENCES

Anderson, T. W. (1963). The use of factor analysis in the statistical analysis of multiple time series. *Psychometrika*, *28*, 1–25. doi:10.1007/BF02289543

Åström, K. J., & Murray, R. M. (2010). *Feedback systems: An introduction for scientists and engineers* (Version v2.10c ed.). Princeton, NJ: Princeton University Press.

Boker, S. M., Neale, M. C., & Rausch, J. (2004). Latent differential equation modeling with multivariate multi-occassion indicators. In K. Van Montfort, H. Oud, & A. Satorra (Eds.), *Recent developments on structural equation models: Theory and applications* (pp. 151–174). Dordrecht, Netherlands: Kluwer Academic. doi:10.1007/978-1-4020-1958-6_9

Boker, S. M., & Nesselroade, J. R. (2002). A method for modeling the intrinsic dynamics of intraindividual variability: Recovering the parameters of simulated oscillators in multi-wave panel data. *Multivariate Behavioral Research*, *37*, 127–160.

Bollen, K. A., & Curran, P. J. (2004). Autoregressive latent trajectory (alt) models: A synthesis of two traditions. *Sociological Methods & Research*, *32*, 336–383. doi:10.1177/0049124103260222

Bollen, K. A., & Curran, P. J. (2006). *Latent curve models: A structural equation approach*. Hoboken, NJ: Wiley.

Browne, M. W., & Nesselroade, J. R. (2005). Representing psychological processes with dynamic factor models: Some promising uses and extensions of autoregressive moving average models. In A. Maydeu-Olivares & J. J. McArdle (Eds.), *Contemporary psychometrics: A festschrift for Roderick P. McDonald* (pp. 415–452). Mahwah, NJ: Erlbaum.

Dolan, C. V. (2005). *MKFM6: Multi-group, multi-subject stationary time series modeling based on the Kalman filter* [Computer software manual]. Retrieved from http://users/fmg.uva.nl/cdolan/

Driver, C. C., Oud, J. H. L., & Voelkle, M. C. (2017). Continuous time structural equation modelling with R package ctsem. *Journal of Statistical Software, 77*, 1–35.

Duncan, O. D. (1969). Some linear models for two-wave, two-variable panel analysis. *Psychological Bulletin*, *72*, 177–182. doi:10.1037/h0027876

Durbin, J., & Koopman, S. J. (2001). *Time series analysis by state space methods*. New York, NY: Oxford University Press.

Fahrmeir, L. (1992). Posterior mode estimation by extended Kalman filtering for multivariate dynamic generalized linear models. *Journal of the American Statistical Association*, *87*, 501–509. doi:10.1080/01621459.1992.10475232

Fahrmeir, L., & Wagenpfeil, S. (1997). Penalized likelihood estimation and iterative Kalman smoothing for non-Gaussian dynamic regression models. *Computational Statistics & Data Analysis*, *24*, 295–320. doi:10.1016/S0167-9473(96)00064-3

Fisher, R. A. (1925). Theory of statistical estimation. *Proceedings of the Cambridge Philosophical Society*, *22*, 700–725.

Gates, K. M., Molenaar, P. C. M., Hillary, F. G., & Slobounov, S. (2011). Extended unified SEM approach for modeling event-related fMRI data. *NeuroImage*, *54*, 1151–1158. doi:10.1016/j.neuroimage.2010.08.051

Gilbert, D. P. (2006). *Brief user's guide: Dynamic systems estimation* [Computer software manual]. Retrieved from http://cran.r-project.org/web/packages/dse/vignettes/Guide.pdf

Grewal, M. S., & Andrews, A. P. (2001). *Kalman filtering: Theory and practice using MATLAB* (2nd ed.). New York, NY: Wiley.

Grewal, M. S., & Andrews, A. P. (2008). *Kalman filtering: Theory and practice using MATLAB* (3rd ed.). Hoboken, NJ: Wiley.

Gu, F., Preacher, K. J., Wu, W., & Yung, Y.-F. (2014). A computationally efficient state space approach to estimating multilevel regression models and multilevel confirmatory factor models. *Multivariate Behavioral Research*, *49*, 119–129. doi:10.1080/00273171.2013.866537

Hamaker, E. L., Dolan, C. V., & Molenaar, P. C. M. (2005). Statistical modeling of the individual: Rationale and application of multivariate time series analysis. *Multivariate Behavioral Research*, *40*, 207–233. doi:10.1207/s15327906mbr4002_3

Hamaker, E. L., Kuiper, R. M., & Grasman, R. P. P. P. (2015). A critique of the cross-lagged panel model. *Psychological Methods*, *20*, 102–116. doi:10.1037/a0038889

Hamerle, A., Nagl, W., & Singer, H. (1991). Problems with the estimation of stochastic differential equations using structural equations models. *Journal of Mathematical Sociology*, *16*, 201–220.

Hamilton, J. D. (1994). State-space models. In R. F. Engle & D. L. McFadden (Eds.), *Handbook of econometrics* (Vol. 4, pp. 3039–3080). New York, NY: Elsevier. Retrieved from http://www-stat.wharton.upenn.edu/stine/stat910/

Hartikainen, J., Solin, A., & Särkkä, S. (2011). *Optimal filtering with Kalman filters and smoothers*. Espoo, Finland: Department of Biomedica Engineering and Computational Sciences, Aalto University School of Science.

Harvey, A. C. (1989). *Forecasting, structural time series models and the Kalman filter*. Cambridge, UK: Cambridge University Press.

Helske, J. (2016a). KFAS: Exponential family state space models in R. *Journal of Statistical Software*, 1–38

Helske, J. (2016b). *KFAS: Kalman filter and smoother for exponential family state space models* [Computer software manual]. Retrieved from http://cran.r-project.org/package=KFAS (R package version 1.2.1)

Holmes, E. E., Ward, E. J., & Wills, K. (2012). MARSS: Multivariate autoregressive state-space models for analyzing time-series data. *The R Journal*, *4*(1), 30.

Horn, J. L., & McArdle, J. J. (1980). Perspectives on mathematical/statistical model building (MASMOB) in reseach on aging. In L. W. Poon (Ed.), *Aging in the 1980's: Selected contemporary issues in the psychology of aging* (pp. 503–541). Washington, DC: American Psychological Association. doi:10.1037/10050-037

Jones, R. H. (1993). *Longitudinal data with serial correlation: A state-space approach*. Berlin, Germany: Springer. doi: 10.1007/978-1-4899-4489-4

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Basic Engineering*, *82*, 35–45. doi:10.1115/1.3662552

Kalman, R. E., & Bucy, R. S. (1961). New results in linear filtering and prediction theory. *Transactions of the ASME, Series D, Journal of Basic Engineering*, *83*, 95–108. doi:10.1115/1.3658902

Kim, J., Zhu, W., Chang, L., Bentler, P. M., & Ernst, T. (2007). Unified structural equation modeling approach for the analysis of multisubject, multivariate functional MRI data. *Human Brain Mapping*, *28*, 85–93. doi:10.1002/hbm.20259

King, A. A., Nguyen, D., & Ionides, E. L. (n.d.). Statistical inference for partially observed Markov processes via the R package pomp. *Journal of Statistical Software*, *69*(12), 1–43. doi:10.18637/jss.v069.i12

Kitagawa, G. (1977). An algorithm for solving the matrix equation $X = FXFT + S$. *International Journal of Control*, *25*, 745–753. doi:10.1080/00207177708922266

Koopman, S. J., Shephard, N., & Doornik, J. A. (1999). Statistical algorithms for models in state space using SsfPack 2.2. *Econometrics Journal*, *2*(1), 113–166. doi:10.1111/1368-423X.00023

MacCallum, R. C., & Ashby, F. G. (1986). Relationships between linear systems theory and covariance structure modeling. *Journal of Mathematical Psychology*, *30*, 1–27. doi:10.1207/S15327906MBR3801_5

MATLAB. (2014). Version 8.3 (r2014a). Natick, MA: The MathWorks.

McArdle, J. J., & Epstein, D. (1987). Latent growth curves within developmental structural equation models. *Child Development*, *58*, 110–133.

Molenaar, P. C. M. (1985). A dynamic factor model for the analysis of multivariate time series. *Psychometrika*, *50*, 181–202. doi:10.1007/BF02294246

Molenaar, P. C. M. (2004). A manifesto on psychology as idiographic science: Bringing the person back into scientific psychology, this time forever. *Measurement*, *2*, 201–218. doi:10.1207/s15366359mea0204_1

Muthén, B. O., & Curran, P. J. (1997). General longitudinal modeling of individual differences in experimental designs: A latent variable framework for analysis and power estimation. *Psychological Methods*, *2*, 371–402. doi:10.1037/1082-989X.2.4.371

Muthén, B. O., & Satorra, A. (1995). Technical aspects of Muthén's LISCOMP approach to estimation of latent variable relations with a comprehensive measurement model. *Psychometrika*, *60*, 489–503. doi:10.1007/BF02294325

Muthén, L. K., & Muthén, B. O. (1998–2014). *M*plus *user's guide* (7th ed.). Los Angeles, CA: Muthén & Muthén.

Neale, M. C., Hunter, M. D., Pritikin, J. N., Zahery, M., Brick, T. R., Kirkpatrick, R. M., … Boker, S. M. (2016). OpenMx 2.0: Extended structural equation and statistical modeling. *Psychometrika*, *80*, 535–549. doi:10.1007/s11336-014-9435-8

Neale, M. C., & Miller, M. B. (1997). The use of likelihood-based confidence intervals in genetic models. *Behavior Genetics*, *27*, 113–120. doi:10.1023/a:1025681223921

Nesselroade, J. R., McArdle, J. J., Aggen, S. H., & Meyers, J. M. (2002). Dynamic factor analysis models for representing process in multivariate time-series. In D. S. Moskowitz & S. L. Hershberger (Eds.), *Modeling intraindividual variability with repeated measures data: Methods and applications* (pp. 235–265). Mahwah, NJ: Erlbaum.

Olsson, U., & Bergman, L. R. (1977). A longitudinal factor model for studying change in ability structure. *Multivariate Behavioral Research*, *12*, 221–241. doi:10.1207/s15327906mbr1202_8

Ou, L., Hunter, M. D., & Chow, S. (2017). What's for dynr: A package for linear and nonlinear dynamic modeling in R. *Journal of Statistical Software*.

Oud, J. H. L. (2002). Continuous time modeling of the cross-lagged panel design. *Kwantitatieve Methoden*, *69*, 1–27.

Pek, J., & Wu, H. (2015). Profile likelihood-based confidence intervals and regions for structural equation models. *Psychometrika*, *80*, 1123–1145. doi:10.1007/s11336-015-9461-1

Petris, G. (2010). An R package for dynamic linear models. *Journal of Statistical Software*, *36*(12), 1–16.

Priestley, M., & Subba Rao, T. (1975). The estimation of factor scores and Kalman filtering for discrete parameter stationary processes. *International Journal of Control*, *21*, 971–975. doi:10.1080/00207177508922050

Pritikin, J. N., Hunter, M. D., & Boker, S. M. (2015). Modular open-source software for item factor analysis. *Educational and Psychological Measurement*, *75*, 458–474. doi:10.1177/0013164414554615

Pritikin, J. N., Hunter, M. D., Von Oertzen, T., Brick, T. R., & Boker, S. M. (2017). Many-level multilevel structural equation modeling: An efficient evaluation strategy. *Structural Equation Modeling, 24*, 684–698. doi:10.1080/10705511.2017.1293542

Pritikin, J. N., Rappaport, L. M., & Neale, M. C. (2017). Likelihood-based confidence intervals for a parameter with an upper or lower bound. *Structural Equation Modeling*, *24*, 395–401. doi:10.1080/10705511.2016.1275969

Rauch, H. E., Tung, F., & Striebel, C. T. (1965). Maximum likelihood estimates of linear dynamic systems. *American Institute of Aeronautics and Astronautics Journal*, *3*, 1445–1450. doi:10.2514/3.3166

Singer, H. (2011). Continuous-discrete state-space modeling of panel data with nonlinear filter algorithms. *Advances in Statistical Analysis (Asta)*, *95*, 375–413. doi:10.1007/s10182-011-0172-3

Smith, G. L., Schmidt, S. F., & McGee, L. A. (1962). *Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle*. Washington, DC: National Aeronautics and Space Administration.

Sörbom, D. (1989). Model modification. *Psychometrika*, *54*, 371–384. doi:10.1007/bf02294623

Voelkle, M. C., & Oud, J. H. L. (2015). Relating latent change score and continuous time models. *Structural Equation Modeling, 22*, 366–381. doi:10.1080/10705511.2014.935918

Voelkle, M. C., Oud, J. H., Von Oertzen, T., & Lindenberger, U. (2012). Maximum likelihood dynamic factor modeling for arbitrary N and T using SEM. *Structural Equation Modeling*, *19*, 329–350. doi:10.1080/10705511.2012.687656

West, M., & Harrison, P. (1997). *Bayesian forecasting and dynamic models*. New York, NY: Springer.

Wu, H., & Neale, M. C. (2012). Adjusted confidence intervals for a bounded parameter. *Behavior Genetics*, *42*, 886–898. doi:10.1007/s10519-012-9560-z

Yang, M., & Chow, S. (2010). Using state-space model with regime switching to represent the dynamics of facial electromyography (EMG) data. *Psychometrika*, *75*, 744–771. doi:10.1007/s11336-010-9176-2

APPENDIX A

GENERAL VECTOR AUTOREGRESSIVE MOVING AVERAGE CODE

```
#——————————————————————————
# Load required packages
require(OpenMx)

#——————————————————————————
# Read in data

# obtain these data from the supplemental
  material

ds <- read.table('varma53.txt', header =
  TRUE)
ds <- ds[,1:3]

#——————————————————————————
# General setup

xdim <- 1 #number of latent variables
udim <- 1 #number of covariates
ydim <- 3*xdim #number of observed variables
arlag <- 4 # autoregressive lag
malag <- 3 #0 or 3 # moving average lag
maxlag <- max(c(arlag, malag+1))
blockxdim <- xdim*(maxlag+1)

#manifest and latent variable names
manNames <- paste('y', 1:ydim, sep = '')
latNames <- paste('x', 1:xdim, sep = '')

#——————————————————————————
# Block matrix setup

Ix <- diag(1, nrow = xdim)
Zx <- matrix(0, xdim, xdim)

# list of autoregressive matrices with ran-
  dom start values
Alist <- list()
for(i in 0:(maxlag-1)){
      if(i < arlag){
         Alist[[i + 1]] <- matrix(runif
            (xdim*xdim, -1, 1), xdim, xdim)
      } else {
         Alist[[i + 1]] <- Zx
      }
}

# list of moving average matrices with ran-
  dom start values
Mlist <- list()
for(i in 0:(maxlag-2)){
      if(i < malag){
         Mlist[[i + 1]] <- matrix(runif
            (xdim*xdim, -1, 1), xdim, xdim)
      } else {
         Mlist[[i + 1]] <- Zx
      }
}

# Build AR and MA matrices into needed block
# state space dynamics A matrix
Mfull <- c(list(Ix), Mlist, list(Zx))
Afull <- c(Alist, list(Zx))

arBlock <- do.call(cbind, Afull)
maBlock <- do.call(cbind, Mfull)
idBlock <- cbind(
      diag(1, nrow = (maxlag-1)*xdim),
      matrix(0, nrow = (maxlag-1)*xdim,
         ncol = 2*xdim))
Ablock <- rbind(arBlock, idBlock, maBlock)

#——————————————————————————
# Starting values and more block matrices
# starting factor loadings
# 1s and 0s are assumed fixed
```

```
facpat <- c(1, .9, .9)
startLoads <- matrix(facpat, ydim, xdim)
#startLoads <- matrix(c(facpat, rep(0, 2*
  (ydim-length(facpat))), facpat), ydim,
  xdim)

# starting factor variances and initial
  means
# 1s and 0s are assumed fixed
startVar <- diag(runif(xdim, 1.1, 1.8),
  xdim)
startMean <- matrix(0, xdim, 1)

startResid <- diag(runif(ydim, .1, .8),
  ydim)

Qblock <- matrix(0, nrow = blockxdim,
  ncol = blockxdim)
Qblock[1:xdim, 1:xdim] <- startVar
Cblock <- matrix(0, nrow = ydim,
  ncol = blockxdim)
Cblock[, (blockxdim-xdim+1):blockxdim] <-
  startLoads

cdimnames <- list(manNames,
c(paste(latNames, "_lag", rep(1:maxlag,
  each = xdim), sep = ""), latNames))

#————————————————————————
# OpenMx matrix and model specification

A <- mxMatrix("Full", blockxdim, blockx-
  dim, values = Ablock,
      free = (Ablock! = 0 & Ablock! = 1),
        name = 'A', lbound = –3, ubound = 3)
B <- mxMatrix("Zero", blockxdim, udim,
  name = 'B')
C <- mxMatrix("Full", ydim, blockxdim,
  values = Cblock,
       free = (Cblock! = 0 & Cblock! = 1),
         name = 'C', dimnames = cdimnames,
         ubound = 10)
D <- mxMatrix("Zero", ydim, udim,
  name = 'D')
Q <- mxMatrix("Symm", blockxdim, blockx-
  dim, values = vech(Qblock),
      free = (Qblock! = 0 & Qblock! = 1),
        name = 'Q', lbound = 0, ubound = 10)
R <- mxMatrix("Symm", ydim, ydim,
  values = vech(startResid),
        free = (startResid! = 0),
          name = 'R', lbound = 0)
x0 <- mxMatrix("Full", blockxdim, 1,
  value = startMean, free = FALSE,
  name = 'x0')
P0 <- mxMatrix("Symm", blockxdim, blockx-
  dim, values = diag(1, blockxdim),
  name = 'P0')
u <- mxMatrix("Zero", udim, 1, name = 'u')

#————————————————————————
# Run model

nameVARMA <- paste0('StateSpaceVARMA(',
  arlag, ",", malag, ")")
modelVARMA <- mxModel(
      name = nameVARMA,
      A, B, C, D, Q, R, x0, P0, u,
      mxData(observed = ds, type = 'raw'),
      mxExpectationStateSpace(A = 'A',
         B = 'B', C = 'C', D = 'D', Q = 'Q',
         R = 'R', x0 = 'x0', P0 = 'P0',
         u = 'u'),
      mxFitFunctionML()
)

runVARMA <- mxRun(modelVARMA)

#————————————————————————
# Inspect model

summary(runVARMA)

# Check that the dynamics are stable
# If the complex magnitude (i.e., Mod) of the
  eigenvalues are less than 1
# then the dynamics are stable.
Mod(eigen(mxEval(A, runVARMA))$values)
# Looks good.
```

APPENDIX B

LATENT GROWTH MODEL CODE

```
#————————————————————————
# Load required packages

require(OpenMx)

#————————————————————————
# Read-in and list-ify data

data(myLongitudinalData)
```

```
matplot(t(myLongitudinalData), type = 'l')
dataL <- list()
for(i in 1:nrow(myLongitudinalData)){
      obsI <- unlist(myLongitudinalData[i,])
      dataL[[i]] <- data.frame(y = obsI,
        ID = i, time = 0:4)
}
nSubj <- length(dataL)

#——————————————————————————————
# Create state space matrices for latent
  growth

amat <- mxMatrix("Iden", 2, 2, name = "A")
bmat <- mxMatrix("Zero", 2, 1, name = "B")
clab <- c(NA, "data.time")
cdim <- list(c("y"), c("I", "S"))
cmat <- mxMatrix("Full", 1, 2, FALSE, c(1,
  NA), name = "C",
      dimnames = cdim, labels = clab)
dmat <- mxMatrix("Zero", 1, 1, name = "D")
qmat <- mxMatrix("Zero", 2, 2, name = "Q")
rlab <- "resid"
rmat <- mxMatrix("Diag", 1, 1, TRUE, .2,
  name = "R", labels = rlab)
xlab <- c("meanI", "meanS")
xmat <- mxMatrix("Full", 2, 1, TRUE, c(1,
  1), name = "x0", labels = xlab)
pval <- c(1, .5, 1)
plab <- c("varI", "covIS", "varS")
pmat <- mxMatrix("Symm", 2, 2, TRUE, pval,
  name = "P0", plab, lbound = c(0, NA, 0))
umat <- mxMatrix("Zero", 1, 1, name = "u")

modL <- list(amat, bmat, cmat, dmat, qmat,
  rmat, xmat,
pmat, umat)
modNames <- paste0("Subject", 1:nSubj,
  "LatentGrowth")
expSS <- mxExpectationStateSpace(A = "A",
  B = "B", C = "C", D = "D", Q = "Q", R = "R",
  x0 = "x0", P0 = "P0", u = "u")

#——————————————————————————————
# Create/estimate multisubject model

indivmodels <- list()
for(k in 1:nSubj){
      DataSetForSubjectK <- dataL[[k]]
      indivmodels[[k]] <- mxModel(name =
        modNames[k],
          modL, expSS,
          mxFitFunctionML(),
          mxData(DataSetForSubjectK, type =
            'raw'))
}
multiSubjGrowth <- mxModel(name =
  "MultiGrowth", indivmodels,
mxFitFunctionMultigroup(modNames))

multiSubjGrowthRun <- mxRun(multiSubj-
  Growth)

#——————————————————————————————
# Examine results

summary(multiSubjGrowthRun)

# Kalman Scores for Subject 1
ks <- mxKalmanScores(multiSubjGrowthRun
  $Subject1LatentGrowth)

# These are equal to the empirical Bayes ran-
  dom effects estimates
# from the nlme and lme4 packages
ks$xSmoothed
```