Running head: OPENMX 2.0

OpenMx 2.0: Extended Structural Equation and Statistical Modeling

Michael C. Neale[1], Michael D. Hunter[7], Joshua Pritkin[3], Mahsa Zahery[5], Timothy R. Brick[6], Robert M. Kirkpatrick[1], Ryne Estabrook[4], Timothy C. Bates[2], Hermine H. Maes[1], and Steven M. Boker[3]

[1]Virginia Institute for Psychiatric and Behavioral Genetics, Virginia Commonwealth University

[2]Department of Psychology, University of Edinburgh

[3]Department of Psychology, University of Virginia

[4]Department of Medical Social Sciences, Northwestern University

[5]Department of Computer Science, Virginia Commonwealth University

[6]Department of Human Development and Family Studies, Pennsylvania State University

[7]Department of Psychology, University of Oklahoma

Affiliation

November 6, 2014

Abstract

The new software package OpenMx 2.0 for structural equation and other statistical modeling is introduced and its features are described. OpenMx is evolving in a modular direction and now allows a mix-and-match computational approach that separates model expectations from fit functions and optimizers. Major backend architectural improvements include a move to swappable open-source optimizers such as the newly-written CSOLNP. Entire new methodologies such as Item Factor analysis (IRT) and State-space modeling have been implemented. New model expectation functions including support for the expression of models in LISREL syntax and a simplified multigroup expectation function are available. Ease-of-use improvements include helper functions to standardize model parameters and compute their Jacobian-based standard errors, access to model components through standard R $ mechanisms, and improved tab completion from within the R Graphical User Interface.

Keywords: Structural Equation Modeling; Path Analysis; Item Response Theory; State Space Modeling; Mixture Distribution; Latent Class Analysis; Behavior Genetics

OpenMx 2.0: Extended Structural Equation and Statistical Modeling

Introduction

The primary aims of this article are to describe the changes that comprise the new release of OpenMx 2.0, present the rationale behind these changes, and to provide a roadmap of what the OpenMx team plans to add in the future. OpenMx is an open source package for what we are calling *extended structural equation modeling* (xSEM) — the idea that a generalization of structural equation modeling (SEM) can incorporate other types of modeling that have traditionally been considered to be beyond the scope of SEM.

Three main aims have driven the development of OpenMx 2.0. The first is the need to enable beginning users to fit new types of models. OpenMx is sufficiently flexible to allow users to build and fit arbitrarily complex models, but these can require significant mathematical, statistical and computing expertise. OpenMx 2.0 expands the range of models that can be specified using path diagramming semantics and thus can be fitted by novice users. Our intent is to make xSEM easier to teach and easier to learn.

Second, in order to enable new models and statistical procedures, we redesigned the way that the objective function is specified. The objective function is the cost function that is minimized in order to obtain optimal parameters for a given model. OpenMx previously had a monolithic objective function, meaning that a single objective function (e.g., `mxRAMObjective`) simultaneously specified how paths and matrices combined (RAM), the fit or cost function (maximum likelihood), and all other optimization details. Upon looking into several different types of optimizers, it became clear that a multipart objective function would enable increased flexibility. OpenMx 2.0 separates the old objective function into three interlocking stages, each of which can be independently

specified. First, OpenMx computes the model's *expectation* for the data, often in the form of an expected covariance and mean. Second, that expectation must be compared to the data using a *fit function*, often a likelihood computation, to determine how well the data fit the model. Finally, the parameters of the model are tuned by an *optimizer* to find the set of parameters that minimize misfit. These three parts are not entirely independent: not all expectations can be operated on by every fit function and optimizer. However, these choices are often quite independent. Accordingly, OpenMx now allows users to specify the expectations of a model in a number of different ways, including path notation, RAM, LISREL or arbitrary matrix algebras. Parameters can be estimated using several combinations of fit functions and optimizers.

Third, OpenMx aspires to be completely free, open, accessible, and extensible. This means that it should both be available to anyone without cost and that the processes it uses to fit models should be available for review and scrutiny by other scientists. Furthermore, it should be easy to access and be easily incorporated into other software packages. Both "Classic" Mx (Neale, Boker, Xie, & Maes, 2003) and OpenMx 1.0 (Boker et al., 2009, 2012) relied upon the state-of-the-art but proprietary package NPSOL (Gill, Murray, Saunders, & Wright, 1986), which optimizes functions subject to linear and non-linear equality or inequality constraints. OpenMx 2.0 adds multiple alternative optimizers, all of which are open-source. CSOLNP, OpenMx's new default optimizer, was independently coded in C++ by M. Zahery & J. Pritkin of the OpenMx team. It is based on RSOLNP (Ghalanos & Theussl, 2012) and the algorithms of Yinyu Ye (Ye, 1987) and is slated for release as a separate R package. CSOLNP shows performance superior to NPSOL for certain classes of problems, especially for models with both continuous and ordinal outcomes. A modified version of NLopt (Johnson, 2010) has also been added to OpenMx 2.0.

The remainder of the article is organized as follows. First, a brief overview of the

extended structural equation modeling (xSEM) framework will be presented; including a description of several features that are maintained from the previous version of OpenMx. This will be followed by details of some of the most important new features in OpenMx 2.0. Next, a few example analyses and scripts are presented in order to illustrate the new functions. To conclude, a roadmap of anticipated future developments to OpenMx is outlined.

Background

OpenMx is open source, meaning that the source code is available for everybody to view, modify, and use. To help organize a community around the project, the OpenMx team maintains a web site `http://openmx.psyc.virginia.edu`, which hosts binary and source versions of the software and several forms of tutorials and reference documentation. Help with OpenMx is available on the web site in discussion forums and a community Wiki. In practice this means that when a problem arises, it is addressed publicly on the community forums. This is in contrast to companies selling closed software, which keep their source code a trade secret (a 'black box') and tend to address any problems privately rather than airing them in public. While the visibility of discussions concerning improving open-source may lead to an impression that it has more problems than closed-source software, studies have reported that this open peer-review process has led to more reliable and higher quality software in comparison to a closed-source approach (e.g., Aberdour, 2007). It is also consistent with the goals of Open Science.

OpenMx is written in a modular fashion, meaning that each section of code operates as independently as possible and is accessed via well-defined interfaces. Perhaps the greatest benefit of modularity is that many programmers can work on the code simultaneously, as long as each module retains the expected behavior of its interface. In the long run, this will enable many more contributors to add to and enhance the

functionality of OpenMx over time, improving and strengthening what is already freely available.

OpenMx runs inside the R statistical programming language and environment (Ihaka & Gentleman, 1996; R Core Team, 2014). The front end is written in the R language, while the computational engine is written in C++, to maximize speed while providing the user with the comprehensive data management, statistics and graphics support afforded by R and its ever growing set of libraries. OpenMx runs on the three major operating systems supported by R: GNU/Linux, Mac OS X, and Microsoft Windows. OpenMx scripts written on one operating system can be run on other operating systems without modification. This platform-independence supports working in today's heterogeneous computing environments, where each researcher on a team may have a different preferred computing platform. Moreover, since OpenMx is free, no license obligations hinder the use of OpenMx on compute clusters or supercomputers where thousands of instances may be run. In addition, OpenMx may be recompiled to take advantage of hardware-optimized numerical software such as the Automatically Tuned Linear Algebra Software (ATLAS) library (Whaley & Dongarra, 1998).

The ability to create expectations for data, to select appropriate fit functions to assess those expectations, and to do the work of parameter optimization lie at the heart of xSEM. The original version of OpenMx implemented a single way to accomplish these three tasks, an important first step. In order to grow, OpenMx 2.0 split these three tasks into modular sections so that users of the software could choose to mix and match them as appropriate to their application problem. In addition, by splitting these tasks into modules and providing an Application Programmer Interface (API) between them, OpenMx opens itself to modification by the community of statistical programmers. The following sections describe this innovative approach to specifying and fitting xSEM models.

Numerical Optimization and Optimizer Selection

Mx and OpenMx 1.0 relied on the optimization package NPSOL (Gill et al., 1986), also implemented as routine E04UCF of the NAg library (The Numerical Algorithms Group (NAG), n.d.). Two features of NPSOL are especially useful: optional derivatives and non-linear constraints on the parameters. Since OpenMx 1.0 allows the user to specify an objective function as an arbitrarily complex matrix algebra, expressions for the first and second derivatives of the fit function are not generally known. NPSOL allows the user to supply any first derivatives that are available, or none at all. The second valuable feature of NPSOL is that it will optimize a fit function subject to linear or non-linear equality or inequality constraints. This capability is accessed within OpenMx through the `mxConstraint()` function. Although valuable, NPSOL has some drawbacks, one of which is that it is commercial and thus source code cannot be distributed along with OpenMx. This prevented distribution of OpenMx via the R library repository CRAN. To address this difficulty, members of the OpenMx who had never seen the NPSOL source developed an equivalently featured optimization package we call CSOLNP. This second optimization engine has proven to be faster and more reliable for certain types of problem. For instance, analyses in which the function precision is not close to machine precision, as is the case with normal theory maximum likelihood applied to ordinal data (i.e., threshold models), appear substantially more robust using CSOLNP than NPSOL. Finally, facilitated by the modular expandable nature of OpenMx as a platform, the open-source NLopt family of optimizers are now selectable (Johnson, 2010).

As an example of optimizer switching, consider the case of a user comparing the performance of the CSOLNP and NPSOL optimizers on a problem. After loading the OpenMx library using `library(OpenMx)`, a user could select the CSOLNP optimizer using the line `mxOption(NULL, "Default optimizer", "CSOLNP")`. After building the model `myModel`, the line `myFit <- mxRun(myModel)` would fit the model using the CSOLNP

optimizer. The output of `summary(myFit)` shows the details of the fitted model and reports which optimizer was used. By entering the command `mxOption(NULL, "Default optimizer", "NPSOL")`, the NPSOL optimizer could be enabled. Running the same model again using `newFit <- mxRun(myModel)` would re-run the same model, now using the NPSOL optimizer.

Alongside new optimizer choices, OpenMx 2.0 supports parallel computation via Symmetric Multiprocessing (SMP) on multicore systems. Users can choose how many processor cores to utilize via the `mxOption()` "Number of threads." By default, OpenMx will use one fewer than the total number of real and virtual cores available. For example, on a Macbook Pro with dual processors each dual-core with hyperthreading, OpenMx will use 8-1=7 threads. On a Linux cluster using the Portable Batch System (PBS; Henderson, 1995), the following script meta-command `#PBS -l nodes=1:ppn=11` will request 11 threads. Such parallel computation can give valuable speed increases, most obviously for raw data analysis (in which the likelihood of each vector of observations in the dataset is computed for every trial set of parameter estimates), confidence interval computations, and threshold models. Our experience is that the benefits of using more than half the available cores, although faster in absolute terms, yields less performance per thread. What are known as *embarrassingly parallel* tasks such as bootstrapping or simulation (in which the same model is fitted many times) can be handled using scripting and batch systems such as Snowfall (Schmidberger et al., 2009) and PBS.

Expectation Functions

In OpenMx 2.0, "expectation" refers to the model-implied expectation about the data. The expectation function contains precisely enough information to simulate data from the model. Typically this would include predicted covariances, with predicted means and thresholds as necessary, and the format of the data (e.g., which columns contain

ordinal data). It does not include information about the method of fitting or the optimizer.

OpenMx 1.0 implemented a single *objective* function, e.g., `mxMLObjective()`. Under OpenMx 2.0, the model-implied expectation and the fit function are specified separately. This allows greater flexibility for composing the desired combination of expectation and fit, without multiplying the two which would yield a huge number of combination functions. Essentially, it creates a modular architecture and consistent API for adding new expectation and fit functions.

What does this mean for the user of OpenMx? Suppose one wanted to obtain maximum likelihood parameter values for a model whose expected covariance matrix is "expCov". In OpenMx 1.0 this had been specified as:

```
mxMLObjective(covariance = "expCov")
```

The equivalent statements in an OpenMx 2 script now are:

```
mxExpectationNormal(covariance = "expCov")
mxFitFunctionML()
```

OpenMx 2.0 is backward compatible and warns the user of the new way of specifying the model while continuing to correctly run the old objective functions. The warning gives copy-pastable updated code and help links for more information. Scripts are not *forward* compatible: a script written to use OpenMx 2.0 expectation and fit functions will not run under any versions of OpenMx 1.0. A brief discussion is presented below for four of the new expectation functions: mxExpectationRAM, mxExpectationNormal, mxExpectationLISREL, and mxExpectationStateSpace. Three of the four expectation functions to be described are capable of generating an expectation for any combination of ordinal, binary, and continuous indicators (the state space expectation currently allows only continuous data). So, before we begin the discussion of expectation functions it will be helpful to understand a little about how OpenMx 2.0 treats binary and ordinal data.

A Note on Binary and Ordinal Data

A binary or ordinal data column in a data frame is considered to be a non-continuous indicator of a continuous underlying distribution that meets the same assumptions as would a continuous data column. The mapping between the values x of this underlying distribution and the binary/ordinal value of each row of data is handled by a thresholding function. For a binary variable, the function specifies that any value of the underlying variable below the threshold will be indicated by a 0 and any value above the threshold will be indicated by a 1. In general, any ordinal variable taking on k values must have $k-1$ thresholds. Ordinal values are assigned to continuous underlying values using the function:

$$f(x) = \begin{cases} 0, & x < T_0 \\ m, & T_{m-1} < x < T_m \\ k-1, & x > T_{k-2} \end{cases} \tag{1}$$

Where T_m represents threshold m, numbered in the range $[0, 1, ..., k-1)$.

These thresholds are specified as a *threshold matrix* with one column per ordinal or binary variable in the data, and one threshold per row. If columns differ in how many thresholds they require, the largest number of rows is used. As a result, the threshold matrix for a model with twelve binary variables would be size 1×12, while the threshold matrix for a model with ten binary variables and two ordinal variables, each with four levels would be size 3×12. Within each column, thresholds must be strictly increasing. The threshold matrix can be defined as an MxMatrix or MxAlgebra. If an element of a threshold matrix is free to be estimated, it should be set to a reasonable starting value given the particulars of the data set.

RAM Expectation

OpenMx continues to allow users to create models using the `type="RAM"` argument to `MxModel()` and then specifying single-headed or double-headed paths between variables as defined by the Reticular Action Modeling (RAM) system for SEM (McArdle & McDonald, 1984; McArdle & Boker, 1990; Boker, McArdle, & Neale, 2002). In this case, the 3 RAM matrices (A, S, and F) and a (row) vector of means (M) are automatically generated and added to the model before it is fit. A means vector is only added if the means are specified by an `mxPath()` statement. When using RAM, the expectation for the covariance is defined to be $F(I-A)^{-1}S(I-A)^{-1'}F'$, and the expectation for the means is defined as $M(I-A)^{-1'}F'$. If one or more columns of data are ordinal or binary, thresholds can be quickly specified using the `mxThreshold()` command. The user may instead directly specify any of the three RAM matrices, the means vector, or the thresholds matrix as either an MxMatrix or MxAlgebra object if so desired. In that case, the automatically generated versions will be overridden.

If the `type="RAM"` argument to `mxModel()` is used, an `MxExpectationRAM` and an `MxFitFunctionML` are also automatically generated. The `MxFitFunctionML` is described in a later section. Either or both of these two functions can be supplied by a user to override the automatic defaults. Overall, we designed this change so that new or occasional users of OpenMx 1.0 who are used to specifying RAM-style models will not need to learn anything new in order to download and use OpenMx 2.0. More advanced users can override, e.g., the fit function used to optimize a RAM-style model. If the user decides not to use `type="RAM"`, it is possible to use `mxExpectationRAM()` to manually specify the expectation. In this case, nothing is created automatically, leaving the user free to specify any RAM model she wishes.

Normal Expectation

The `mxExpectationNormal()` function allows one to specify model-implied expected covariances and means under the assumption of multivariate normality. The `covariance=` and `means=` arguments point to MxMatrix or MxAlgebra objects that define the model-implied expected covariance and means respectively. When `mxExpectationNormal()` is used, all matrices and algebra that define these model-implied expectations must be created by the user. If any variables are ordinal or binary, a thresholds matrix is also required.

As an example use of `mxExpectationNormal()`, a user might use the `type="RAM"` argument to `mxModel()`, along with `mxPath()` statements to have OpenMx create A, S, and F matrices. The user could use these automatically-created matrices along with other user-supplied matrices in an mxAlgebra statement to define a new model-implied expected covariances in a way different from the default RAM specification. For instance, this trick is often used in time-delay embedding models where an additional derivative estimation filter matrix will be created and incorporated into the model-implied covariance calculation.

LISREL Expectation

A new feature for OpenMx 2.0 is the ability to directly specify LISREL type models (Jöreskog & Van Thillo, 1972). The user can provide information on the 13 LISREL matrices (Hunter, 2012; $\Lambda_x, \Lambda_y, B, \Gamma, \Phi, \Psi, \Theta_\delta, \Theta_\epsilon, \Theta_{\delta\epsilon}, \tau_x, \tau_y, \kappa$, and α). Importantly, the user can also specify any valid subset of these matrices. The first subset of LISREL that may be of interest is a model without means. This removes the τ_x, τ_y, κ, and α means vectors and only models the covariance data. The second subset of some utility is an endogenous-only model. This is a 4-to-6-matrix specification with no exogenous matrices, consisting only of $\Lambda_y, B, \Psi, \Theta_\epsilon$ and optionally the means τ_y and α. The third subset is an

exogenous-only model. This is a measurement model with no structural components. The exogenous-only subset is a factor model, a 3-to-5-matrix specification consisting of $\Lambda_x, \Phi, \Theta_\delta$ and optionally the means τ_x and κ.

When using the LISREL expectation function, special cases of the LISREL model are not automatically created or intuited from the user's input. For instance, consider a model in which all exogenous variables are manifest. This could be accomplished by setting the Λ_x matrix to the identity matrix and fixing the Φ and Θ_δ matrices to zero matrices. Manifest exogenous variables could then impact the endogenous variables through the Γ regression coefficients matrix. There is no built-in special case for this situation. However, matrix constructor functions for identity matrices, zero matrices, and unit matrices are available in OpenMx 2.0. While path specification of LISREL models is not currently supported, it is a planned future feature. Note that the syntax of the LISREL software from Scientific Software International (Jöreskog & Sörbom, 1999) is not supported. While a motivated user could write a parser that would read LISREL syntax and output the OpenMx equivalent, the OpenMx team itself has no plans to write or support such a tool.

The LISREL expectation is another example of the modular implementation of OpenMx. The same data processing and fit function options exist for LISREL as they do for RAM and Normal expectations. The same code, in fact, that computes ordinal, joint ordinal-continuous, or continuous data maximum likelihood is used by RAM and LISREL. Because of the modularity, separate missing data handling or ordinal data handling is not needed. The LISREL expectation was simply written as a new expectation module.

Further improvements were made to the LISREL expectation based on findings from the RAM expectation. In particular, one optimization that is used by RAM was also found to be applicable to LISREL. The RAM expectation requires the inversion of $I - A$, but uses some heuristics to determine if the expression would be faster to calculate as a

finite but equal series: $(I - A)^{-1} = I + A + A^2 + A^3 + \ldots$, up to some natural number p (Boker et al., 2002). In many cases, A is nilpotent of low order p such that $A^p = 0$. In a RAM path diagram, $p - 1$ is the length of the longest string of single headed arrows that can be constructed connected head to tail. Often, models have no variables that are both predictors and predicted. In that case $p = 2$ and so $(I - A)^{-1} = I + A + A^2 + \ldots$ reduces to $I + A$, which can be much faster to compute. The LISREL expectation requires a similar inverse, $I - B$, where B is also often nilpotent of low order, so the same speed-up is applied.

The LISREL expectation can help when teaching with OpenMx. Introductory SEM textbooks (e.g. Maruyama, 1998) are frequenly written using the LISREL notation and perspective on modeling. It is useful to distinguish between the measurement model and the structural model, and between exogenous variables and endogenous variables. The LISREL notation facilitates these distinctions of model components while the RAM notation does not. It can be difficult to describe some models in the LISREL perspective. McArdle's dual change score model is an example of the ability of RAM notation and path diagrams to create novel SEMs (see McArdle & Hamagami, 2001; Hamagami & McArdle, 2007; King et al., 2006; Chow, Grimm, Filteau, Dolan, & McArdle, 2013). It has been said that LISREL teaches you the rules of SEM while RAM teaches you how to break them. OpenMx's flexible architecture permits users to take advantage of the strengths of both specifications as needed.

The example below demonstrates factor modeling with means using the LISREL expectation function.

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

```
factorMeans <- mxMatrix("Zero", 1, 1, name="Kappa", dimnames=list("F1", NA))
xIntercepts <- mxMatrix("Zero", nvar, 1, name="TauX", dimnames=list(varnames, NA))
factorLoadings <- mxMatrix("Full", nvar, 1, TRUE, .6, name="LambdaX",
         labels=paste("lambda", 1:nvar, sep=""), dimnames=list(varnames, "F1"))
factorCovariance <- mxMatrix("Diag", 1, 1, FALSE, 1, name="Phi")
xResidualVariance <- mxMatrix("Diag", nvar, nvar, TRUE, .2, name="ThetaDelta",
         labels=paste("theta", 1:nvar, sep=""))
liModel <- mxModel(model="LISREL Factor Model",
factorMeans, xIntercepts, factorLoadings, factorCovariance, xResidualVariance,
mxExpectationLISREL(LX="LambdaX", PH="Phi", TD="ThetaDelta", TX="TauX", KA="Kappa"),
mxFitFunctionML(),
mxData(demoOneFactor, "raw")
)
liRun <- mxRun(liModel)
summary(liRun)
```

State space modeling

State space modeling is a new feature to OpenMx (Hunter, 2014), and to the authors' knowledge, new to any SEM program. Many programs exist for state space modeling (e.g. MATLAB, 2014; Koopman, Shephard, & Doornik, 1999; Dolan, 2005; Browne & Zhang, 2010; Petris & Petrone, 2011; Petris, 2010) but OpenMx may be the first to directly support state space modeling in a flexible, extendable SEM environment. State space models are discrete-time linear dynamical systems of variables that are not measured perfectly. They are similar in many respects to stochastic differential equations (e.g Arminger, 1986). State space models (SSMs, also called dynamic factor analysis, process factor analysis, latent autoregressive time series) form recursive relationships

where the latent variables at one time are related to the same variables later in time. These are called autoregressive dynamics. The latent variables in turn produce measured variables contemporaneously, like an ordinary factor analysis model.

Using notation from engineering, a state space model can be written as

$$\vec{x}_{t+1} = A\vec{x}_t + B\vec{u}_t + \vec{q}_t \tag{2}$$

$$\vec{y}_t = C\vec{x}_t + D\vec{u}_t + \vec{r}_t \tag{3}$$

where there are l latent variables, m manifest variables, and c covariates. Then $\vec{x}_t$ is a $l \times 1$ vector of the latent states, $\vec{u}_t$ is a $c \times 1$ vector of observed covariates/inputs, $\vec{q}_t$ is a $l \times 1$ vector of dynamic noise, $\vec{y}_t$ is a $m \times 1$ vector of observed outputs, $\vec{r}_t$ is a $m \times 1$ vector of observation noise, A is an $l \times l$ matrix of autoregressive dynamics, B is an $l \times c$ matrix of covariate/input effects on the state, C is an $m \times l$ matrix of factor loadings, D is an $m \times c$ matrix of covariate/input effects on the observation, Q is the $l \times l$ covariance matrix of the dynamic noise $\vec{q}_t$, and R is the $m \times m$ covariance matrix of the observation noise $\vec{r}_t$. Equation 2 is called the state equation, and Equation 3 is called the output equation. The noise vectors $\vec{q}_t$ and $\vec{r}_t$ have zero mean, are assumed to be uncorrelated with each other, uncorrelated with themselves at other times, and uncorrelated with the observations $\vec{y}_t$. The model forms a recursive relation of the multivariate mean and variance of the next row of data given the mean and variance of the current row of data. Thus, SSMs are Gaussian models just like factor analysis and other SEMs (see also Roweis & Ghahramani, 1998).

For researchers familiar with SEM, putting the same state space equations in the notation of Mplus/LISCOMP might make them more understandable.

$$\vec{\eta}_{i+1} = B\vec{\eta}_i + \Gamma\vec{x}_i + \vec{\zeta}_i \tag{4}$$

$$\vec{y}_i = \Lambda\vec{\eta}_i + K\vec{x}_i + \vec{\epsilon}_i \tag{5}$$

For comparison the Mplus/LISCOMP SEM equations for the structural and measurement models are nearly identical

$$\vec{\eta}_i = B\vec{\eta}_i + \Gamma\vec{x}_i + \vec{\zeta}_i \tag{6}$$

$$\vec{y}_i = \Lambda\vec{\eta}_i + K\vec{x}_i + \vec{\epsilon}_i \ , \tag{7}$$

the only difference being that SSMs predict forward in time ($\vec{\eta}_{i+1}$) while normal SEM equations are contemporaneous ($\vec{\eta}_i$). The measurement model of a SEM is identical to the output equation of the state space model. The structural model of a SEM only differs from the state equation of the state space model in the subscript of the left-hand-side of the equation. In SEM models the influence of the latent variables for time step i is on the same time step i, whereas the SSM models the influence of the latent variables for time step i is on the *next* time step $i + 1$.

The modeled relationship from one unit to another makes time series modeling quite natural for SSMs. Models with large numbers of time points (> 1000) and large numbers of latent and manifest variables (> 20) can be specified and estimated both quickly and conveniently. The primary difference in data structure is that in SSMs different rows represent distinct *times* rather than people. Multiple people are incorporated via multigroup modeling in which each person is a group. This kind of clustering may lead some to think SSM may be related to multilevel models — and they are. Some multilevel SEMs can be fit as SSMs (Gu, Preacher, Wu, & Yung, 2014).

Similar to the addition of the LISREL expectation, the modularity of OpenMx aided in the development of the state space expectation. The state space expectation extends the normal expectation by computing its own expected covariance and means using a Kalman filtering approach. Once these expectations are computed, however, the same `MxFitFunctionML` is used to compute the likelihood of the data as is used for a RAM or LISREL expectation.

Examples are included in the help(mxExpectationStateSpace), including dynamic

factor analysis, standard factor analysis, and using covariates. A short example is provided below to illustrate the use of `mxExpectationStateSpace()`.

```
library(OpenMx)
data(demoOneFactor)
nvar <- ncol(demoOneFactor)
varnames <- colnames(demoOneFactor)
ssModel <- mxModel(model="State Space Manual Example",
    mxMatrix("Full", 1, 1, TRUE, .3, name="A"), #Autoregressive parameters
    mxMatrix("Zero", 1, 1, name="B",
       dimnames=list("F1", "U1")), # covariates predict latent variables
    mxMatrix("Full", nvar, 1, TRUE, .6, name="C",
       dimnames=list(varnames, "F1")), # factor loadings matrix
    mxMatrix("Zero", nvar, 1, name="D",
       dimnames=list(varnames, "U1")), # covariates predict observed vars
    mxMatrix("Diag", 1, 1, FALSE, 1, name="Q"), # latent residual covariance
    mxMatrix("Diag", nvar, nvar, TRUE, .2, name="R"), # observed residual cov.
    mxMatrix("Zero", 1, 1, name="x0"), # initial latent estimate
    mxMatrix("Diag", 1, 1, FALSE, 1, name="P0"), # variance of initial estimate
    mxMatrix("Zero", 1, 1, name="u"), # covariates
    mxData(observed=demoOneFactor, type="raw"),
    mxExpectationStateSpace(A="A", B="B", C="C", D="D", Q="Q",
                            R="R", x0="x0", P0="P0", u="u"),
    mxFitFunctionML()
)
ssRun <- mxRun(ssModel)
summary(ssRun)
```

Fit Functions

OpenMx defines a fit function as a method of comparing model-implied expectations to data and reducing that comparison to a number or vector. Each model can only have one fit function, but any model can contain other models (called *child models*) and all of these child models need not have the same fit function. In fact, the parent model for a set of child models quite often has a different fit function than its children.

Four fit functions from OpenMx 2.0 are described below: mxFitFunctionML for maximum likelihood fit; mxFitFunctionAlgebra for optimizing the value of an algebra; mxFitFunctionMultigroup for aggregating the fit functions of child models; and mxFitFunctionRow for optimizing a user-specified algebra fit function on each row of data.

Maximum Likelihood Fit Function

The `mxFitFunctionML()` function computes $-2\log(\text{likelihood})$ of data given the present values of the free parameters and the expectation function selected for the model. Most often, this expectation function will be `mxExpectationNormal()` or `mxExpectationRAM()` and `mxFitFunctionML()`, which will return $\sum -2\log(\text{likelihood})$ over the data. However, in some instances it is useful to obtain a vector of likelihoods, one for each row in the data. In this case, one may call `mxFitFunctionML(vector=TRUE)` and the likelihood of each row in the data set will be reported. An `mxAlgebra()` is then needed to process the individual likelihoods into the negative twice log-likelihood of the sample.

The results of running a model are generally examined by using the summary function, which reports the results of an OpenMx optimization in a summarized format. However, the result returned by a model's fit function can also be directly accessed. This is illustrated in the following trivial example, which calculates the mean and variance of a variable $x1$, shows a summary, and then prints just the resulting fit function value:

```
library(OpenMx)
```

```
data(demoOneFactor)
trivialModel <- mxModel(model="Trivial Model", type="RAM",
manifestVars="x1",
mxPath(from="x1", arrows=2, free=TRUE, labels="varX1"),
mxPath(from="one", to="x1", arrows=1, free=TRUE, labels="meanX1"),
mxFitFunctionML(),
mxData(demoOneFactor, type="raw")
)
trivialModelOut <- mxRun(trivialModel)
summary(trivialModelOut)
trivialModelOut$output$fit
```

Note that we suggest you use `trivialModelOut$output$fit` to access the result of fit functions. While `mxFitFunctionML(vector=TRUE)` still populates the previously used slot called "Minus2LogLikelihood" in order to maintain compatibility with previous code, this is not true for the other fit functions since they might not be calculating a likelihood.

Algebra Fit Function

While the other fit functions in OpenMx require an expectation function for the model, the `mxFitFunctionAlgebra()` function uses the mxAlgebra referenced in its argument `algebra=` as the function to be minimized. To evaluate an algebra fit function, place the following objects in a MxModel object: an mxFitFunctionAlgebra and the MxAlgebra and MxMatrix as shown in the trivial example below that adjusts two free parameters until they are equal to one another:

```
library(OpenMx)
trivialModel2 <- mxModel(model="Optimize to equality",
mxMatrix("Full", nrow=1, ncol=1, free=TRUE, values=5, name="A"),
```

```
mxMatrix("Full", nrow=1, ncol=1, free=TRUE, values=10, name="B"),
mxAlgebra((A-B)^2, name="SqrDistance"),
mxFitFunctionAlgebra(algebra="SqrDistance")
)
trivialModel2Out <- mxRun(trivialModel2)
mxEval(SqrDistance, trivialModel2Out)
mxEval(A, trivialModel2Out)
mxEval(B, trivialModel2Out)
trivialModel2Out$output$fit
```

Note that the result of the mxAlgebra named "SqrDistance" is now what is located in `trivialModel2Out$output$fit`, which is not a likelihood at all!

The `mxFitFunctionAlgebra()` function is frequently used when aggregating the fit results from multiple child models together. When this aggregation applies to multi-group modeling, OpenMx 2.0 has a new fit function that simplifies this process: `mxFitFunctionMultigroup`.

Multigroup Fit Function

The fit function `mxFitFunctionMultigroup()` facilitates multi-group modeling. OpenMx implements multiple group models by enclosing these component groups as child models within a parent model. The parent model evaluates a joint fit function optimizing across all its child models. Prior to OpenMx 2.0, the user specified a joint fit `mxAlgebra` value (typically the sum of likelihoods of the component models), and added an `mxAlgebraObjective` to the parent model so it could be optimized against this criterion. In OpenMx 2.0, the mxFitFunctionMultigroup conveniently wraps the creation of this algebra and fit function: the user simply lists the name of each of child model as arguments to the mxFitFunctionMultigroup. In addition to being easier to use, it more

clearly communicates the user's intention. For example, the presence of `mxFitFunctionMultigroup()` makes it possible to identify the component models that comprise the multi-group model. This is useful for automatically generating reference models.

Row-wise Fit Function

A second extension of `mxFitFunctionAlgebra()` is `mxFitFunctionRow()` which evaluates an mxAlgebra on each row of data, gathers the results, and then uses another user-specified mxAlgebra to reduce those row results down to a single measure of fit. The mxFitFunctionRow function is important because it allows specification of an algebraic fit function that can differ across rows. For example, handling missing data may sometimes require mxFitFunctionRow rather than mxFitFunctionAlgebra. Full information maximum likelihood can be implemented using `mxFitFunctionRow()` as a teaching tool to better understand the procedure.

Other New Features

Compute plans

Like most software for fitting statistical models, OpenMx 1.0 performed a fixed sequence of steps when asked to optimize a model. OpenMx 2.0 introduces the MxCompute object which is a "compute plan" that communicates to OpenMx's computational engine specifically what the user wants to do while optimizing a given model. The default compute plan is identical to the actions that are performed by OpenMx 1.0. That is, the model is optimized by a non-linear optimizer, likelihood-based confidence intervals are estimated, and, if requested, the Hessian matrix is estimated by finite differences. OpenMx 2.0 breaks these steps into separately configurable pieces using a small domain-specific language.

Implementing the MxCompute object was necessary in order to address optimization algorithms such as expectation maximization (EM) which does not follow the computational actions in OpenMx 1.0. Compute plans are extremely flexible and open up the door to a wide variety of optimization algorithms and model types. One of these is Item Factor Analysis.

Item Factor Analysis (IFA)

With Item Factor Analysis, OpenMx takes a large step forward into the world of Item Response Theory (IRT) models. While we will only briefly mention its existence in the current article, the interested reader can find out much more about the implementation of IFA in OpenMx from Pritikin and colleagues' recent article describing the method and how it is used in practice (Pritikin, Hunter, & Boker, in press). Recently implemented speedups in the IFA routines in OpenMx 2.0 have made it one of the fastest software routines available for IRT.

The IFA module offers a novel model specification language that is well suited to programmatic generation and manipulation of models. Modular organization of the source code facilitates the easy addition of item models, item parameter estimation algorithms, optimizers, test scoring algorithms, and fit diagnostics all within an integrated framework. The availability of both IFA and structural equation modeling in the same software is a step toward the unification of these two methodologies (Pritikin et al., in press).

Improved Output and Reporting

Summary statistics have been enhanced in several ways. Support for computing the saturated and independence models for models with raw data is now included, enabling fit indices which depend on them to be reported in summary. This aids the user in reporting 'absolute' fit indices such as the Comparative Fit Index (Bentler, 1990) or the Tucker-Lewis Index (Tucker & Lewis, 1973). When RMSEA is reported, it now also has

its 95 percent confidence interval to allow users to examine approximate fit. Additionally, the summary method now has two distinct printing styles, one verbose and the other more limited. The limited style is the default. The verbose style may be useful for debugging and a more in-depth look at model results. The validity of these indices in the presence of missing data is an area for further research.

Standardized parameter estimates for RAM models have been reported in OpenMx since the 1.0 release. These were a transformation of the original parameters based on the estimated expected covariance matrix. However, this transformation itself has some degree of uncertainty because it is based on imperfectly estimated parameters. This same transformation was applied to the calculate the standardized standard errors reported in OpenMx 1.0. However, these standardized standard errors did not reflect the uncertainty in the transformation. The new mxStandardizeRAMpaths function solves this problem by accurately accounting for the uncertainty in the transformation.

In addition, OpenMx 2.0 now supports automated computation of reference models (saturated and independence models) for many raw and covariance data models via the `mxRefModels()` function. Reference models are needed to compute fit indices such as CFI and TLI. The output from the `mxRefModels()` function can now be given to summary to obtain more complete fit indices for raw data models. The mxRefModels function works for covariance data, raw continuous data, ordinal data, raw joint ordinal/continuous data with RAM, LISREL, and Normal expectations. For certain data structures, the saturated and independence models created by mxRefModels may not be appropriate. These cases are outlined in the help page for this function. Broadly speaking, when multiple variables are actually multiple instances of the same variable (e.g. at a different time, or in a different twin), the saturated and independence models may need to be modified to reflect these identities. Reference models can also be computed for IFA using the `mxRefModels()` function.

A common problem for complex models, is choosing suitable starting values for estimated parameters. OpenMx 2.0 features a new function, `mxTryHard()` which makes multiple attempts at running a model until the optimizer reports a converged estimate. Between attempts, it uses the most recent set of parameter estimates as start values on the next attempt, moving them within the parameter space to help find values from which the model will converge.

OpenMx 1.0 implemented an accessor function `mxEval()` which provided access portions of a model's output. However, we found that the user community was more inclined to access model components directly, using a somewhat confusing mixture of @ and $ signs. OpenMx 2.0 has regularized its accessors so that only the $ sign is needed, just as is common practice in the R language. Although the @ signs still work (in almost all cases), we *very strongly encourage* people to remove @ signs from their scripts and only use $ signs. This allows the OpenMx team to maintain backwards compatibility as we move into the future and insures that the scripts that written today will still run on future versions of OpenMx.

External Libraries

As OpenMx has become more established, an ecosystem of specialized packages has become available. Examples include: Cheung's meta-SEM package (Cheung, 2014) supporting complex SEM-based meta-analyses; von Oertzen & Brandmeier's graphical SEM package Ωnyx, which generates OpenMx scripts from path diagrams (von Oertzen, Brandmaier, & Tsang, in press); and user help function libraries such as Bates' umx and umx.twin (Bates, 2013), which provide easy-to-use helper functions for common tasks in OpenMx. The OpenMx development team welcomes new add-on packages and is open to providing support to developers of them.

Planned Developments

The OpenMx team are actively working on multilevel models and the least squares family of fit functions. Some simple multivariate multilevel models can be estimated with OpenMx 2.0 by using the state space expectation function (as in Gu et al., 2014). But the OpenMx development team is working towards a much more general solution that would accommodate cross-classified models as well as large and complex data. We are still working on syntax for general purpose multilevel models that is both comprehensive and simple to understand.

In the development version of OpenMx (which currently must be downloaded and compiled from the trunk archive on the OpenMx website), Unweighted Least Squares and some Weighted Least Squares models can be estimated with a version of `mxFitFunctionWLS`. While this does pass the tests we have run, we want to run a wider variety of tests as well as make sure that user-supplied identification constraints are easy-to-use and comprehensive before we release this fit function.

Summary

Like its 1.0 release, OpenMx 2.0 is a full-featured, open-source xSEM package that runs on most operating systems. In addition to the variety of models that could be specified in previous versions, the 2.0 release adds support for more complex models and modern computing architectures, and lays the groundwork for additional feature development. OpenMx's focus on being both open source and modular allows for both transparency and extensions to handle new models and paradigms. Thus, OpenMx can exist not just as a stand-alone xSEM package, but a vehicle by which SEM can be learned and integrated into other statistical methods. As always, if the interested reader wishes to suggest future developments or participate in discussions about them, the place to start is the forums section of the OpenMx website at `http://openmx.psyc.virginia.edu`.

References

Aberdour, M. (2007). Achieving quality in open-source software. *Software, IEEE*, *24*(1), 58–64.

Arminger, G. (1986). Linear stochastic differential equation models for panel data with unobserved variables. *Sociological Methodology*, *16*, 187-212. Retrieved from `http://www.jstor.org/stable/270923`

Bates, T. C. (2013). umx: A help package for structural equation modeling in openmx [Computer software manual]. Edinburgh, UK. Retrieved from `http://github.com/tbates/umx/` (version 0.6)

Bentler, P. M. (1990). Comparative fit indexes in structural models. *Psychol Bull*, *107*, 238-246.

Boker, S., McArdle, J. J., & Neale, M. C. (2002). An algorithm for the hierarchical organization of path diagrams and calculation of components of covariance between variables. *Structural Equation Modeling*, *9*(2), 174–194.

Boker, S., Neale, M., Maes, H., Wilde, M., Spiegel, M., Brick, T., ... Fox, J. (2009). *OpenMx: Multipurpose software for statistical modeling.* (University of Virginia, Department of Psychology, Box 400400, Charlottesville, VA 22904. http://openmx.psyc.virginia.edu)

Boker, S., Neale, M., Maes, H., Wilde, M., Spiegel, M., Brick, T., ... Fox, J. (2012). *OpenMx: Multipurpose software for statistical modeling, version 1.2.* (University of Virginia, Department of Psychology, Box 400400, Charlottesville, VA 22904. http://openmx.psyc.virginia.edu)

Browne, M., & Zhang, G. (2010). DyFA 3.00 user guide. Retrieved from `http://quantrm2.psy.ohiostate.edu/browne/`

Cheung, M. W.-L. (2014). metaSEM: Meta-analysis using structural equation modeling [Computer software manual]. Retrieved from

`http://courses.nus.edu.sg/course/psycwlm/Internet/metaSEM/` (R package version 0.9-0)

Chow, S.-M., Grimm, K. J., Filteau, G., Dolan, C. V., & McArdle, J. J. (2013). Regime-switching bivariate dual change score model. *Multivariate Behavioral Research*, *48*(4), 463–502.

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

Ghalanos, A., & Theussl, S. (2012). RSOLNP: general non-linear optimization using augmented lagrange multiplier method [Computer software manual]. (R package version 1.14.)

Gill, P. E., Murray, W., Saunders, M. A., & Wright, M. H. (1986). *User's guide for NPSOL (version 4.0): A FORTRAN package for nonlinear programming.* (Tech. Rep.). Department of Operations Research, Stanford University.

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*(2), 119–129.

Hamagami, F., & McArdle, J. J. (2007). Dynamic extensions of latent difference score models. In S. M. Boker & M. J. Wenger (Eds.), *Data analytic techniques for dynamical systems.* Lawrence Erlbaum Associates.

Henderson, R. L. (1995). Job scheduling under the Portable Batch System. In D. G. Feitelson & L. Rudolph (Eds.), *Job scheduling strategies for parallel processing* (pp. 279–294). Berlin-Heidelberg: Springer-Verlag.

Hunter, M. D. (2012, July 9-12). *The addition of LISREL specification to OpenMx.* Lincoln, NE. (Presented at the 2012 Annual International Meeting of the Psychometric Society)

Hunter, M. D. (2014, May 22-25). *Extended structural equations and state space models*

when data are missing at random. San Francisco, CA. (Presented at the 2014 Annual Meeting of the Association for Psychological Science)

Ihaka, R., & Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, *5*(3), 299–314.

Johnson, S. G. (2010). *The NLopt nonlinear-optimization package.* R package http://ab-initio.mit.edu/nlopt.

Jöreskog, K. G., & Sörbom, D. (1999). *Lisrel 8: User's reference guide.* Lincolnwood, IL: Scientific Software International.

Jöreskog, K. G., & Van Thillo, M. (1972). LISREL: A general computer program for estimating a linear structural equation system involving multiple indicators of unmeasured variables. *ETS Research Bulletin Series*, *DOI: 10.1002/j.2333-8504.1972.tb00827.x*.

King, L. A., King, D. W., McArdle, J. J., Saxe, G. N., Doron-LaMarca, S., & Orazem, R. J. (2006). Latent difference score approach to longitudinal trauma research. *Journal of Traumatic Stress*, *19*, 771-785.

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.

Maruyama, G. M. (1998). *Basics of structural equation modeling.* Thousand Oaks, CA: Sage Publications.

MATLAB. (2014). *Version 8.3 (R2014a).* Natick, Massachusetts: The MathWorks Inc.

McArdle, J. J., & Boker, S. (1990). *Rampath.* Hillsdale, NJ: Lawrence Erlbaum.

McArdle, J. J., & Hamagami, F. (2001). Linear dynamic analyses of incomplete longitudinal data. In L. Collins & A. Sayer (Eds.), *New methods for the analysis of change* (p. 137-176). Washington, DC: American Psychological Association.

McArdle, J. J., & McDonald, R. P. (1984). Some algebraic properties of the Reticular Action Model for moment structures. *British Journal of Mathematical and*

Statistical Psychology, *37*, 234–251.

Neale, M. C., Boker, S. M., Xie, G., & Maes, H. H. (2003). *Mx: Statistical modeling.* (VCU Box 900126, Richmond, VA 23298: Department of Psychiatry. 6th Edition)

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

Petris, G., & Petrone, S. (2011). State space models in R. *Journal of Statistical Software*, *41*(4), 1-25.

Pritikin, J. N., Hunter, M. D., & Boker, S. (in press). Modular open-source software for Item Factor Analysis. *Educational and Psychological Measurement*.

R Core Team. (2014). R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria. Retrieved from `http://www.R-project.org/`

Roweis, S., & Ghahramani, Z. (1998). A unifying review of linear Gaussian models. *Neural Computation*, *11*(2), 305-345.

Schmidberger, M., Morgan, M., Eddelbuettel, D., Yu, H., Tierney, L., & Mansmann, U. (2009). State-of-the-art in parallel computing with R. *Journal of Statistical Software*, *47*(1).

The Numerical Algorithms Group (NAG). (n.d.). *The NAG Library http://www.nag.com.* Oxford, United Kingdom.

Tucker, L. R., & Lewis, C. (1973). A reliability coefficient for maximum likelihood factor analysis. *Psychometrika*, *38*, 1–10.

von Oertzen, T., Brandmaier, A., & Tsang, S. (in press). Structural equation modeling with ωnyx. *Structural Equation Modeling: A Multidisciplinary Journal*.

Whaley, R. C., & Dongarra, J. J. (1998). Automatically tuned linear algebra software. In *Proceedings of the 1998 ACM/IEEE conference on supercomputing* (pp. 1–27).

Ye, Y. (1987). *Interior algorithms for linear, quadratic, and linearly constrained non-linear programming.* Unpublished doctoral dissertation, Department of ESS,

Stanford University.

Author Note

The authors gratefully acknowledge funding from the National Institutes of Health, specifically grants R01-DA022989 (PI Boker), R37-DA018673 and R25-DA026119 (PI Neale). Thanks are also due to a large group of beta-testers, including but not limited to: Mike W.-L. Cheung (Cheung, 2014), Charles Driver, Dorothy Bishop, Greg Carey, Pascal Deboeck, Emilio Ferrer, Christopher Hertzog, Kevin Grimm, Ken Kelley, Matthew Keller, Jean-Philippe Laurenceau, Gitta Lubke, John J. McArdle, Sam McQuillin, Sarah Medland, William Revelle, Michael Scharkow, James Steiger, Melissa Sturge-Apple, and Theodore Walls.