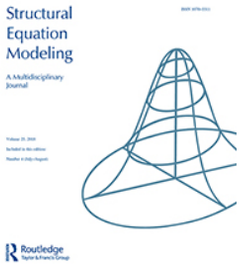
Routledge
Taylor & Francis Group



Structural Equation Modeling: A Multidisciplinary Journal

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

MplusAutomation: An R Package for Facilitating Large-Scale Latent Variable Analyses in M*plus*

Michael N. Hallquist & Joshua F. Wiley

To cite this article: Michael N. Hallquist & Joshua F. Wiley (2018) *MplusAutomation*: An R Package for Facilitating Large-Scale Latent Variable Analyses in M*plus*, Structural Equation Modeling: A Multidisciplinary Journal, 25:4, 621-638, DOI: 10.1080/10705511.2017.1402334

To link to this article: https://doi.org/10.1080/10705511.2017.1402334

Structural Equation Modeling: A Multidisciplinary Journal, 25: 621–638, 2018

ISSN: 1070-5511 print / 1532-8007 online
DOI: https://doi.org/10.1080/10705511.2017.1402334

TEACHER'S CORNER

MplusAutomation: An R Package for Facilitating Large-Scale Latent Variable Analyses in M*plus*

Michael N. Hallquist[1] and Joshua F. Wiley[2]
[1]*The Pennsylvania State University*
[2]*Monash University*

MplusAutomation is a package for R that facilitates complex latent variable analyses in M*plus* involving comparisons among many models and parameters. More specifically, *MplusAutomation* provides tools to accomplish 3 objectives: to create and manage M*plus* syntax for groups of related models; to automate the estimation of many models; and to extract, aggregate, and compare fit statistics, parameter estimates, and ancillary model outputs. We provide an introduction to the package using applied examples including a large-scale simulation study. By reducing the effort required for large-scale studies, a broad goal of *MplusAutomation* is to support methodological developments in structural equation modeling using M*plus*.

Keywords: latent variable analysis, Monte Carlo study, M*plus*, R

Several packages within the R language (R Core Team, 2015) provide excellent open-source tools for fitting structural equation models, including lavaan (Rosseel, 2012) and OpenMX (Neale et al., 2016). Nevertheless, proprietary structural equation modeling (SEM) programs such as LISREL (Jöreskog & Sörbom, 2015), AMOS (Arbuckle, 2014), and M*plus* (Muthén & Muthén, 2017) enjoy widespread use for a variety of reasons including ease of use, specialized modeling facilities, and users' familiarity. M*plus* is among the most popular SEM programs because of its relatively simple programming syntax, support for advanced analyses, and commitment to making contemporary methodological developments accessible to applied researchers. Specialized SEM software such as M*plus*, however, does not offer a complete statistical programming language that supports detailed secondary analyses (e.g., distributions of fit statistics across simulated replications) or the preparation of publication-quality figures and tables. Here, we describe *MplusAutomation*, an R package that facilitates the creation, management, execution, and interpretation of large-scale latent variable analyses using M*plus*.

The *MplusAutomation* package extends the flexibility and scope of latent variable analyses using M*plus* by overcoming some of its practical limitations. First, however, we wish to highlight the strengths of M*plus* that have led to its prominence in both theoretical and applied SEM research. Foremost is that M*plus* is currently the most comprehensive SEM program, providing facilities for Bayesian, multilevel, exploratory, mixture, item response theory, longitudinal, non-Gaussian, and complex survey extensions of SEM. The creators of M*plus* regularly make substantive contributions to the SEM literature (e.g., Asparouhov & Muthén, 2014; Muthén & Asparouhov, 2012) and consistently implement these new methods in M*plus*. The programming syntax of M*plus* is reasonably intuitive for applied researchers, using natural language such as "ON" to represent the regression of *Y* on *X* or "WITH" to represent the covariation of *X* with *Y*. Finally, M*plus* offers highly optimized implementations of computationally expensive methods such as bootstrapped confidence intervals and multidimensional numerical integration.

Correspondence should be addressed to Michael N. Hallquist, Department of Psychology, Pennsylvania State University, 141 Moore Building, University Park, PA 16802. E-mail: michael.hallquist@psu.edu

Color versions of one or more of the figures in the article can be found online at www.tandfonline.com/hsem.

Despite these strengths, M*plus* is not well suited to run large batches of models (e.g., a set of twin models for many candidate genes), although recent versions provide some batch-related facilities such as tests of measurement invariance. The root limitation is that, like most SEM programs, M*plus* relies on a one input–one output approach in which syntax for a single model produces an output file containing parameter estimates, fit statistics, and other model details. Consequently, to estimate a variety of models across data sets or simulated replications, users must produce a unique syntax file for each instance. Likewise, M*plus* stores output in text format,[1] requiring users to identify and extract relevant values manually from each output file. Indeed, much of the underlying code of *MplusAutomation* uses text extraction methods to convert M*plus* outputs into R-compatible data structures. In addition to facilitating analyses of model outputs, *MplusAutomation* allows researchers to leverage the outstanding capabilities of R to produce figures and tables for SEM-based research (Wickham & Grolemund, 2017). R is also a leading language for developing literate programming documents that support reproducible research practices (Gandrud, 2015).

INTRODUCTORY EXAMPLE: CONFIRMATORY FACTOR ANALYSIS

To introduce the *MplusAutomation* package, we begin with a simple confirmatory factor analysis (CFA) of Fisher's well-known iris data set containing flower measurements for three species. Assuming one has installed M*plus* and R, the first step is to install and load the *MplusAutomation* package. To improve the format of output, we also suggest installing the *texreg* package. Installation is accomplished using `install.packages`, which only needs to be done once, and loading is accomplished using `library`, which must be done each time R starts.

```
> install.packages("MplusAutomation")
> install.packages("texreg")
> library(MplusAutomation)
> library(texreg)
```

If the data are stored as a `data.frame object` in R, the next step is to export them `using prepareMplusData`, which saves an M*plus*-compatible tab-delimited file and provides an M*plus* syntax stub with variable names and the data file specification.

```
> prepareMplusData(iris[, 1:4], "iris.dat")
TITLE: Your title goes here
DATA: FILE = "iris.dat";
VARIABLE:
NAMES = Sepal_Length Sepal_Width Petal_Length Petal_Width;
MISSING=.;
```

Next, we build on the syntax stub to create a complete M*plus* input file. Here, we specify a single-factor CFA of the four variables, and save the model in the file `cfa_example.inp`.

```
TITLE: Single Factor CFA;
DATA: FILE = "iris.dat";
VARIABLE:
NAMES = Sepal_Length Sepal_Width Petal_Length Petal_Width;
MISSING=.;

MODEL:
F BY Sepal_Length Sepal_Width Petal_Length Petal_Width;

OUTPUT: CINTERVAL;
```

Finally, we run the model in M*plus* using `runModels`, read the results back into R using `readModels`, and print a summary of the parameters and fit indexes using `screenreg`.

```
> runModels("cfa_example.inp")
> cfares <- readModels("cfa_example.out")
> screenreg(cfares, cis = TRUE, single.row = TRUE,
+ summaries = c("BIC", "CFI", "SRMR"))
```

Confidence intervals (if available from M*plus*) are requested by `cis = TRUE`, and `single.row = TRUE` displays confidence intervals next to, not below, parameter estimates. The output of the `screenreg` function is provided in Table 1.

APPLIED EXAMPLE: COMPARING RELATIVE EVIDENCE OF CONTINUOUS VERSUS CATEGORICAL LATENT STRUCTURE

To illustrate how *MplusAutomation* facilitates latent variable modeling using M*plus*, we describe model estimation and comparison of CFA and latent class analysis (LCA). See Table 2 for a description of the core *MplusAutomation* functions illustrated in this example. For the purpose of demonstration, we first generated data in M*plus* using a two-class latent class model with higher item probabilities for two of six binary items in one class and lower probabilities for the other four items.[2] In applied studies of classification and taxonomy, researchers are often

[1] We note, however, that recent versions of M*plus* store some estimates in a hierarchical data format (HDF) file amenable to direct importation by R or other software.

[2] M*plus* syntax and output are available in the *MplusAutomation* repository at https://github.com/michaelhallquist/MplusAutomation/tree/master/examples/lca_cfa_example.

TABLE 1
Parameter Estimates and Fit Statistics for a One-Factor Confirmatory Factor Analysis

Parameter	Estimate
SEPAL_LENG<-F	1.00 [1.00; 1.00][a]
SEPAL_WIDT<-F	−0.35 [−0.44; −0.26][a]
PETAL_LENG<-F	2.78 [2.44; 3.11][a]
PETAL_WIDT<-F	1.03 [0.89; 1.16][a]
SEPAL_LENG<-Intercepts	5.84 [5.71; 5.97][a]
SEPAL_WIDT<-Intercepts	3.06 [2.99; 3.13][a]
PETAL_LENG<-Intercepts	3.76 [3.48; 4.04][a]
PETAL_WIDT<-Intercepts	1.20 [1.08; 1.32][a]
F<->F	0.45 [0.30; 0.59][a]
SEPAL_LENG<->SEPAL_LENG	0.24 [0.18; 0.29][a]
SEPAL_WIDT<->SEPAL_WIDT	0.13 [0.11; 0.16][a]
PETAL_LENG<->PETAL_LENG	−0.34 [−0.50; −0.18][a]
PETAL_WIDT<->PETAL_WIDT	0.11 [0.07; 0.14][a]
BIC	879.55
CFI	0.92
SRMR	0.10

Note. BIC = Bayesian information criterion; CFI = comparative fit index; SRMR = standardized root mean square residual.

[a]Indicates that the confidence interval does not include zero.

interested in whether a latent construct such as depression is best conceptualized as dimensional, categorical, or a hybrid (e.g., see Hallquist & Pilkonis, 2012). Although there are deeper conceptual issues in resolving latent structure, scientists often depend on nonnested model comparisons using criteria such the Akaike's information criterion (AIC) to adjudicate among candidate models (Hallquist & Wright, 2014; Markon & Krueger, 2006).

A typical analytic pipeline to compare dimensional versus categorical structure is (a) estimate LCAs with an increasing number of classes; (b) resolve the number of latent classes that best characterizes the data using the bootstrapped likelihood ratio test (BLRT) or model selection criteria such as AIC (Nylund, Asparouhov, & Muthén, 2007); (c) estimate a one-factor CFA and consider the possibility of multidimensional solutions; (d) inspect model fit and modification indexes for the CFA to ensure that no serious misspecification is present; and (e) decide between categorical and dimensional representations on the basis of interpretability, model selection criteria (e.g., AIC), and relative evidence (e.g., Bayes factors). Typically, this pipeline requires creating input files for each model, running each model individually in M*plus*, and visually parsing the output to extract key features such as item probabilities. These might be copied manually into a spreadsheet program, summarized, and formatted for publication.

To reduce the burden of managing multiple syntax files, *MplusAutomation* leverages the fact that a project often shares many features across model variants. For example, CFA and LCA models of the same data set minimally share the variable names, data structure, and the overall structure of M*plus* syntax. Rather than copying and pasting M*plus* code with adaptations for each model variant, *MplusAutomation* allows users to set up a single template file that contains shared syntax across models, as well as unique specifications for each variant. After setting up a template file, the `createModels` function outputs a set of M*plus* syntax files. Importantly, if an aspect of the project changes (e.g., adding additional variables to the data), updating the template file and rerunning `createModels` propagates changes to each M*plus* syntax file.

As in our initial CFA example, data can be converted to M*plus* format using `prepareMplusData`. Next, one creates a template file specifying all model variants to be tested. Appendix A shows a template file that generates syntax files for fitting a one-factor CFA and two- to five-class LCAs (five total models). Most of the code is conventional M*plus* syntax, whereas template-related syntax is encapsulated in double square brackets. Template files begin with an *init* section (Lines 1–15) that defines one or more *iterator* variables. Analogous to cells in a factorial design, `createModels` loops over these iterators to create unique syntax files for each combination. For example, if one were interested in fitting two- to five-class LCAs in six independent data sets, then the *init* section would specify `iterators = classes dataset;`, and the numeric levels of each iterator would be: `classes = 2:5; dataset = 1:6;`. If additional attributes of each level are important for creating model syntax, these can be specified by defining a variable that is yoked to a given iterator. For example, the model iterator in the example is associated with `modelnames#model` (Line 4), which defines the descriptions of each model that are used to name the files (e.g., "CFA"). The *init* section also specifies the name of the syntax file (Line 13) and the output directory (Line 14). These are helpful to keep code and results organized, especially when there are many model variations. The variables defined in the *init* section are used in the M*plus* syntax section (Lines 16–33) to substitute attributes dynamically such as the title of the model or the number of classes.

Finally, template files can include code conditionally by placing it between opening and closing tags, distinguished by a forward slash (e.g., `[[model == 1]]` and `[[/model == 1]]`, respectively). The M*plus* syntax between these tags is included when the condition within the brackets is true. In the LCA example, *MplusAutomation* adds the `CLASSES` statement only when the model iterator is above one (Line 22) because the first model is a single-factor CFA.

After setting up the template file (stored as a text file), the user then generates all relevant M*plus* input files using createModels:

```
> createModels("C:/templateExample.txt")
```

This function loops through commands in the specified template file, creates M*plus* input files according to the filename variable in the `[[init]]` section, and places them within the specified `outputDirectory`. After running `createModels`, we suggest inspecting some of the M*plus* syntax files for accuracy. Next, to estimate all of the models for the project, one uses the `runModels` command:

TABLE 2
Summary of Major *MplusAutomation* Functions in the Order of a Typical Data Analysis Workflow

Function Name	Description	Example
createModels	Processes an M*plus* template file and creates a group of related M*plus* input syntax files. Definitions and examples for the template language are provided in the online *MplusAutomation* vignette (http://cran.r-project.org/web/packages/MplusAutomation/vignettes/Vignette.pdf).	`createModels("template_file.txt")`
prepareMplusData	Converts an R *data.frame* object into a tab-delimited file (without header) to be used in an M*plus* input file. The corresponding M*plus* syntax, including the data file definition and variable names, is printed to the console or optionally to an input file.	`study5 <- foreign::read.spss("s5data.sav", to.data.frame = TRUE)` `prepareMplusData(df = study5, filename = "study5.dat", keepCols = paste0("x", 1:25))`
mplusObject	For inline integration of M*plus* and R, this function creates an object that contains all M*plus* input syntax sections needed to specify a single model, as well as a field for the data set. Primarily used for specifying and running M*plus* models within R using *mplusModeler*.	`example1 <- mplusObject(TITLE = "MPG regression", MODEL = "mpg ON wt;", usevariables = c("mpg", "hp"), rdata = mtcars)`
mplusModeler	Creates all files necessary to run an M*plus* model specified by an *mplusObject*. When *run* = 1, the model is also run in M*plus* and the output is returned as an *mplus.model* object following the format used by *readModels*. When *run* is greater than 1, the number indicates how many nonparametric bootstrap replications to run.	`outm <- mplusModeler(example1, dataout = "mtcars.dat", modelout = "mtcarsreg.inp", run = 1L)`
runModels	Runs a group of *M*plus models (.inp files) located within a directory or nested within subdirectories. Optionally, one can specify a single .inp file to estimate a specific model.	`runModels("/Users/mh/mplus_demo", recursive = TRUE, replaceOutfile = "modifiedDate")`
runModels_Interactive	Provides an interactive interface for selecting a batch of models to run using *runModels*.	`runModels_Interactive()`
readModels	Extracts all supported sections of one or more M*plus* output files, including model parameters and summary statistics. Data are extracted into *mplus.model* objects based on the *list* data type in R. If the *target* is a single output file, then a single *mplus.model* object is obtained. If the target is a directory, a list of objects (named according to the output file) is returned.	`knownclassModels <- readModels(target = "/Users/mh/mplus_demo", recursive = TRUE, filefilter = ".*knownclass")`
SummaryTable	Output summary statistics for one or more models in tabular format. The user can specify which summaries are displayed, the criterion on which models are sorted, and the format of the output. Output types include HTML, LaTeX, and Markdown, as well as on-screen display.	`SummaryTable(knownclassModels, type = "html", sortBy = "BIC")`
compareModels	Outputs summary statistics and parameters for two models side-by-side for detailed comparison. Inputs are *mplus.model* objects obtained from *readModels*. A chi-square difference test comparing nested models is also provided using *diffTest = TRUE*.	`compareModels(mlist[["h2_config.out"]], mlist[["h2_strong.out"]], diffTest = TRUE, showFixed = FALSE)`
paramExtract	Subset a *data.frame* containing M*plus* parameters (from the *$parameters* field of *mplus.model* objects) to obtain parameters of only a specified type (e.g., regression coefficients or factor loadings).	`paramExtract(mlist[["h2_config.out"]]$parameters$unstandardized, "loading")`

```
> runModels(directory = "C:/templateExample", recursive = TRUE)
```

Specifying `recursive = TRUE` runs input files in all subdirectories of `C:/templateExample`. Depending on the number and complexity of the models to execute, batch estimation could take considerable time. Users can check the log file created by `runModels` (named "Mplus Run Models.log" by default) to follow the progress on the batch.

Next, the `readModels` function imports various parts of M*plus* output files into R data structures for further analysis, visualization, or examination. To import all models pertaining to our example, we specify the directory containing the output files:

```
> lcacfa <- readModels("templateExample", recursive = TRUE)
```

One can generate tables of model summaries and fit indexes by passing the object from `readModels` to the `SummaryTable` function, which displays a model table to the screen, or outputs it as HTML, Markdown, or LaTeX:

```
> SummaryTable(lcacfa)
```

##	Title	LL Parameters	AIC	AICC	BIC
## 1	2-class	LCA -3600.324	13 7226.649	7227.018	7290.450
## 2	3-class	LCA -3596.928	20 7233.856	7234.714	7332.012
## 3	4-class	LCA -3592.598	27 7239.197	7240.753	7371.706
## 4			12 7245.741	7246.057	7304.634
## 5	5-class	LCA -3589.627	34 7247.253	7249.719	7414.117

As anticipated, the model summary table indicates that the two-class LCA model best fits the data according to both AIC and Bayesian information criterion (BIC). Finally, as described next, one can display detailed comparisons of parameters between two models using the compareModels function. For example, one could compare the two- and three-class LCA models in detail:

```
> compareModels(lcacfa$templateExample.
LCA2.LCA2.out,
+ lcacfa$templateExample.LCA3.LCA3.out)
```

In summary, even with this relatively basic example of a latent structure analysis, *MplusAutomation* saves the user from manually creating related syntax files and automates the unattended execution of model estimation. Having the results available in R also facilitates many other tasks, such as producing tables and figures.

Researchers more proficient in R can accomplish the same analyses in a more R-centric framework using the `mplusObject` and `mplusModeler` functions. The benefit of this approach is that R and M*plus* syntax can be integrated inline, providing greater programmatic control of the models using R syntax for flow control and conditional logic. Here, we provide an R-centric version of the same analyses accomplished earlier using a template file, `createModels`, `runModels`, and `readModels`. First, if the data to be modeled are stored in a `data.frame`, `mplusObject` creates an R representation of an M*plus* input file and data set:

```
> m.cfa          <- mplusObject(
+                 TITLE = "Confirmatory Factor Analysis",
+                 VARIABLE = "categorical = u1 - u6;",
+                 ANALYSIS = "estimator = mlr;",
+                 MODEL = "factor BY u1-u6;",
+                 OUTPUT = "TECH1; TECH8;",
+                 PLOT = "TYPE = PLOT3;",
+                 usevariables = colnames(d),
+                 rdata = d)
```

The function has arguments corresponding to the major sections of M*plus* input files (`e.g., TITLE, VARIABLE, MODEL`). Aside from providing the data set and names of the variables to use, it is not necessary to specify the variable names or data set output location, which *MplusAutomation* handles automatically. The `mplusModeler` function exports syntax and data files, calls M*plus* to estimate the model, and reads the results back into R:

```
> m.cfa.fit <- mplusModeler(m.cfa,
"cfa_example.dat", run = TRUE)
```

An advantage of the mplusObject approach is that one can easily update and modify an existing model during interactive model building. This is accomplished using the update function with a formula syntax that follows R conventions (e.g., regression model specification). Updates occur by section, and omitting a section keeps it the same as the original model. To replace an entire section, use the syntax `[SECTION NAME] = ~ "new syntax"`. To amend a section, adding additional commands to the existing specification, use the syntax `[SECTION NAME] = ~ . + "extra syntax"`. For example, adding syntax is helpful for including residual covariances based on modification indexes, while other parts of the model remain the same. Because updated models are stored in new R objects, users can maintain a history of the interactive modeling process. That is, rather than repeatedly editing the same model code and reestimating, which loses a record of previous versions, each model can be easily documented and retained. The update approach also helps to identify how a model differs from previous variants, as only changes are specified.

Using the update function, we can modify the syntax for a one-factor CFA to estimate LCAs:

```
> m.lca   <- lapply(2:5, function(k) {
+         body <- update(m.cfa,
+         TITLE = ~ "Latent Class Analysis",
+         VARIABLE = as.formula(sprintf("~ . +
            'classes = c(%d);'", k)),
+         ANALYSIS = ~ . + "type = mixture; starts =
            250 25;",
+         MODEL = ~ "%OVERALL%",
+         OUTPUT = ~ . + "TECH14;")
+
+         mplusModeler(body, sprintf("lca_%
            d_example.dat", k), run = TRUE)
+ })
```

This code uses the `lapply` function to iterate through two- to five-class models, storing all results in the `m.lca object`. Each time, R replaces the `TITLE` and `MODEL` sections, expands the `ANALYSIS` and `OUTPUT` sections, and reuses the variables and data from the CFA model. The VARIABLE section dynamically specifies the number of classes across models.

Regardless of whether M*plus* models are set up using a template file with `createModels`, or are specified inline using an `mplusObject`, the `readModels` function imports results into R as `mplus.model` objects with a predictable structure (see Table 3). For example, one can easily generate a high-quality, customized graph (e.g., using the popular *ggplot2* package; Wickham, 2016) of the model-estimated item frequencies from the two-class LCA model, as depicted in Figure 1 (R code for this plot is provided in Appendix B).

DETAILED MODEL COMPARISON USING *MPLUSAUTOMATION*

To build and validate structural equation models, researchers often compare related model variants that differ on potential

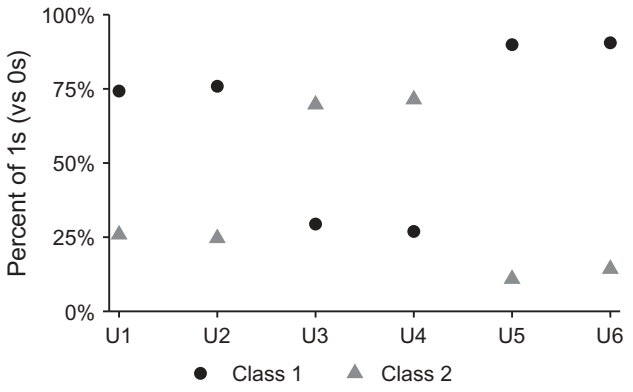


FIGURE 1 Model-estimated proportions of item responses for a two-class latent class model estimated by M*plus*. Symbols (circle and triangle) denote latent class and the binary indicators U1 through U6 are arrayed along the *x* axis.

explanatory variables, such as additional predictors or mediating variables (Kline, 2015). Likewise, one might wish to compare the relative evidence of nested or nonnested models (Merkle, You, & Preacher, 2016), such as in measurement invariance testing. Parameter estimates typically change when a correlated explanatory variable is added to the model. In larger SEMs this leads to the challenge of identifying which estimates are substantially altered by the addition of parameters or variables. Moreover, when comparing large models, keeping track of which parameters are unique to a given variant can be difficult (e.g., the transition from variances to residual variances when a variable is made endogenous to the model).

The compareModels function in *MplusAutomation* provides a detailed comparison of two models in a side-by-side format, both for model summaries and parameters. For nested models, the function can compute a model difference test based on the log-likelihood or model χ^2 by passing `diffTest = TRUE`. Furthermore, users can specify whether to display parameters that are equal or different between models, based on parameter estimates, *p* values, or both. One can specify the definition for relative equality using the `equalityMargin` argument. Finally, using the show argument, users can specify which aspects to compare, including summaries, unique parameters, equal, and unequal parameters. An example of a detailed CFA model comparison with the addition of a residual covariance parameter is provided in Appendix C.

CONDUCTING A MONTE CARLO STUDY USING *MPLUSAUTOMATION*

Monte Carlo simulation studies are an important tool in latent variable research to characterize the performance of models under a variety of conditions where the data generation processes and latent structure are known. Simulation studies in SEM research often report summary information about model fit, parameter estimate bias, and parameter coverage (i.e., the efficiency with which the technique recovers the true parameters) across simulated replication data sets. Simulation studies have been conducted for numerous purposes in the SEM literature, such as the performance of missing data methods (Enders & Bandalos, 2001) or the efficiency of Bayesian versus maximum likelihood multilevel SEM for estimating cluster-level effects (Hox, Van De Schoot, & Matthijsse, 2012). *M*plus provides excellent functionality for simulating data from a variety of latent variable models, as well as basic Monte Carlo analyses including average fit, bias, coverage, and mean squared error (e.g., Muthén & Muthén, 2002). Moreover, M*plus* can combine parameter estimates across replications generated internally or by other data simulation software.

Using *MplusAutomation* for Monte Carlo studies extends M*plus* by providing detailed information about each replication and leveraging R to manage large-scale studies where replications and analyses vary along several dimensions (e.g., sample size, data generation process, type of model misspecification, or the amount of missingness). Because of the one input–one output approach of M*plus*, simulation studies that require the extraction and organization of information from thousands of output files quickly become intractable using manual output parsing (e.g., copy and paste into a spreadsheet). Extending the latent structure example earlier, we show how to conduct a basic Monte Carlo study of categorical versus dimensional structure using *MplusAutomation*. More specifically, the example focuses on how corrupting data generated from a latent class model affects model selection decisions. This idea builds on the literature on the robustness of model-based clustering to scatter observations not drawn from the ground truth model (Maitra & Ramler, 2009). Following Markon and Krueger (2006), the relative evidence for categorical versus dimensional representations of a set of psychometric indicators can be resolved using model selection criteria such AIC and BIC. Thus, if data are simulated from a latent class model and fit by a CFA model, model selection criteria should provide greater evidence for the LCA than CFA. Such comparisons can inform a better understanding of how best to represent the taxonomy of psychological constructs and psychopathology, for example.

To illustrate the power of *MplusAutomation* and M*plus* for Monte Carlo studies, we simulated replications from a three-class LCA[3] with five uncorrelated normally distributed indicators (i.e., a diagonal covariance matrix) and equally sized classes:

$$f(y) = \sum_{k=1}^{K} P(k) f(y|\mu_k,\ \Sigma_k)$$

$$y_k \sim N(\mu_k,\ \Sigma_k)$$

[3] Technically this is a latent profile model because the indicators are continuous, but we use the term LCA to encompass latent class models with both categorical and continuous variables.

TABLE 3
Description of M*plus* Output Imported Into an R Data Structure by the *readModels* Function

Element	Description	Common Subelements	Example of Selected Contents
$ input	M*plus* model input syntax parsed into a list of relevant sections and subsections.	$ title	`"Example Mplus model"`
		$ data	`$file: "example.dat"`
		$ variable	`$names: "grp y1 y2 y3 x1 x2 x3"` `$auxiliary: "(e) x1 x2 x3"` `$useobservations: "grp EQ 2"` `$classes: "c (3)"`
		$ analysis	`$type: "MIXTURE RANDOM"` `$starts: "1500 200"` `$algorithm: "INTEGRATION"`
		$ model	`[1] "%OVERALL" [2] "y1 WITH y2;"`
		$ plot	`[1] "TYPE = PLOT3;"`
		$ output	`[1] "TECH1 TECH10 TECH14 STANDARDIZED;"`
$ warnings	A *list* of input syntax and model estimation warnings.		`[[1]] "STANDARDIZED (STD, STDY, STDYX) options are not available for TYPE = RANDOM."`
$ errors	A *list* of input syntax and model estimation errors.		`[[1]] "THE STANDARD ERRORS OF THE MODEL PARAMETER ESTIMATES COULD NOT BE COMPUTED. THIS IS OFTEN ..."`
$ sampstat	Sample statistics (e.g., sample mean and covariance) produced by OUTPUT: SAMPSTAT.	$ means	`BSIGSI NEGAFFEC HAMD IIP_PD1` `1 0.862 23.425 15.113 1.474`
		$ covariances	`BSIGSI NEGAFFEC HAMD IIP_PD1` `BSIGSI 0.654 NA NA NA` `NEGAFFEC 5.981 86.120 NA NA` `HAMD 5.109 50.134 78.896 NA` `IIP_PD1 0.455 5.032 3.636 0.769`
		$ correlations	`W1TOB W2TOB W3TOB` `W1TOB 0.993 NA NA` `W2TOB 0.855 0.859 NA` `W3TOB 0.827 0.773 0.831`
$ covariance_coverage	Proportion of sample having observed data for each bivariate combination of variables.		
$ summaries	A *data.frame* of summary information and fit statistics for the model, often including log-likelihood, AIC, BIC, RMSEA, TLI, and CFI. For mixture models with TECH11 or TECH14 enabled, tests of *k*-class versus *k* – 1-class solutions are also included.	$ Mplus.version	`"7.1"`
		$ Estimator	`"MLR"`
		$ Observations	`240`
		$ Parameters	`20`
		$ LL	`-985.11`
		$ LLCorrectionFactor	`0.239`
		$ AIC	`1992.30`
		$ BIC	`2023.61`
		$ Filename	`"3-class LPA with Aux.out"`
		$ Entropy	`0.818`
$ parameters	A *list* of parameter estimates output by the model, where each element is a *data.frame* typically including the parameter name, estimate, standard error, and *p* value of the parameter. For some models, standardized estimates, R^2, and confidence intervals are also provided.	$ unstandardized	`paramHeader param est se est_se pval LatentClass` `Means Y1 1.052 0.112 9.428 0.000 1` `Means Y2 1.117 0.113 9.877 0.000 1` `Means Y3 -0.834 0.105 -7.965 0.000 1` `Means Y1 -0.916 0.108 -8.483 0.000 2` `Means Y2 2.099 0.186 11.294 0.000 2` `Means Y3 1.791 0.158 11.347 0.000 2`
		$ r2	
		$ stdyx.standardized	
		$ std.standardized	
$ indirect	A *list* of overall and specific indirect effects in the model provided by MODEL INDIRECT	$ overall	`pred outcome summary est se est_se pval` `1 IMPLW1 BPD_I Total 0.029 0.009 3.103 0.002` `2 IMPLW1 BPD_I Total indirect 0.015 0.012 1.230 0.219` `3 IMPLW1 BPD_I Direct 0.014 0.014 0.970 0.332`
		$ specific	`pred intervening outcome est se est_se pval` `1 BPD_I A11XEMOT IMPLW1 -0.002 0.001 -1.744 0.081` `2 BPD_I IMP_I IMPLW1 0.012 0.012 0.966 0.334` `3 BPD_I IMP_S IMPLW1 0.003 0.007 0.359 0.720`

(*Continued*)

TABLE 3
(Continued)

Element	Description	Common Subelements	Example of Selected Contents
$ mod_indices	A *data.frame* of model modification indexes output by the MODINDICES command, when available.		`modV1 operator modV2 MI EPC Std_EPC StdYX_EPC` `ANGRY15 ON BPD16 15.456 0.238 0.218 0.200` `BPD16 BY ANGRY15 15.456 0.238 0.218 0.200` `ANGRY15 ON BPD17 15.856 0.183 0.198 0.182` `BPD17 BY ANGRY15 15.856 0.183 0.198 0.182` `MOODY16 ON BPD15 12.862 0.302 0.269 0.200`
$ savedata_info	A *list* of details about the SAVEDATA output.	$ filename $ fileVarNames $ fileVarFormats $ fileVarWidths $ bayesFile $ bayesVarNames	`chr "02_grandmodel_child.savedata"` `chr [1:57] "A10CSELF" "A11CSELF" "A12CSELF" ...` `chr [1:57] "F10.3" "F10.3" "F10.3" ...` `num [1:57] 10 10 10 ...` `chr "Bresults_M3.dat"` `chr [1:226] "Parameter.1_%G#1%:.MEAN.LIF2"` `"Parameter.2_%G#1%:.MEAN.DEPR2"` `"Parameter.3_%G#1%:.MEAN.LIF1" ...`
$ savedata	A *data.frame* containing data output by the SAVEDATA: FILE command. Examples of data include influence statistics, factor scores, observed data, and latent class membership.		`'data.frame':` `2189 obs. of 57 variables:` `..$ A10CSELF: num [1:2189] 3 10 3 14 ...` `..$ A11CSELF: num [1:2189] 8 5 8 9 ...` `..$ A12CSELF: num [1:2189] 8 4 7 1 ...` `..$ A13CSELF: num [1:2189] 10 17 10 12 ...` `..$ A14CSELF: num [1:2189] 11 17 11 11 ...` `..$ OUTINFL : num [1:2189] 0.019 0.032 0.014 ...` `..$ OUTCOOK : num [1:2189] 0.018 0.032 0.008 ...`
$ residuals	Difference between the observed and model estimated covariances/correlations/ residual correlations. Means are stored as row vectors; covariances are stored as matrices.	$ meanEst $ meanResid $ meanResid.std $ meanResid.norm $ covarianceEst $ covarianceResid $ covarianceResid.std $ covarianceResid.norm $ slopeEst $ slopeResid	`num [1, 1:6] 20.09 230.72 ...` `num [1, 1:6] 0 0 0 0 0 0` `num [1, 1:6] 0 0 0 0 0 0` `num [1, 1:6] 0 0 0 0 0 0` `num [1:6, 1:6] 34.84–5 ...` `num [1:6, 1:6] 0.345–608.313 ...` `num [1:6, 1:6] 0.281–3.628 ...` `num [1:6, 1:6] 0.039–3.628 ...`
$ tech1	A *list* with two elements representing free parameter numbers and starting values. These are stored in vectors and matrices according to the SEM parameterization (e.g., factor loadings are in $lambda).	$ parameterSpecification $ startingValues	`$tau : num [1, 1:6] 7 8 9 10..` `$nu : num [1, 1:6] 0 0 0 0...` `$lambda: num [1:6, 1] 0 1 2 3...` `$theta : num [1:6, 1:6] 0 0 0...` `$alpha : num [1, 1] 0` `$beta : num [1, 1] 0` `$psi : num [1, 1] 6` `$tau : num [1, 1:6] –0.008 ...` `$nu : num [1, 1:6] 0 0 0 0 0...` `$lambda: num [1:6, 1] 1 1 1 1 1...` `$theta : num [1:6, 1:6] 1 0 0 0...` `$alpha : num [1, 1] 0` `$beta : num [1, 1] 0` `$psi : num [1, 1] 0.05`
$ tech3	A *list* with two elements representing the parameter covariance and correlation matrices. Both are square, lower triangular matrices with dimensions matching the number of parameters.	$ paramCov $ paramCor	`1 2 3` `1 1.000 NA NA` `2 –0.007 1.000 NA` `3 –0.776 0.006 1.000`

(*Continued*)

TABLE 3
(Continued)

Element	Description	Common Subelements	Example of Selected Contents
$ tech4	A *list* with three elements for the estimated means, covariances, and correlations for latent variables.	$ latMeansEst	F1 F2 1 0 0
		$ latCovEst	F1 F2 F1 33.652 NA F2 −5.000 0.236
		$ latCorEst	F1 F2 F1 1.000 NA F2 −1.773 1
$ tech7	A *list* of sample statistics (means and covariances) with one element for each class and weighted by posterior class probability.	$ CLASS.1 … $ CLASS.K	$classSampMeans: num [1, 1:4] 5.92 2.75 $classSampCovs: num [1:4, 1:4] 0.252 0.077
$ tech9	A *list* of convergence errors for each replication or bootstrap draw (e.g., for Monte Carlo studies, multiply imputed data, or bootstrapping).	$rep.1 … $rep.K	$ rep172:List of 2 ..$ rep: num 172 ..$ error: chr "NO CONVERGENCE..." $ rep199:List of 2 ..$ rep: num 199 ..$ error: chr "NO CONVERGENCE..."
$ tech12	A *list* of observed, estimated, and residuals for means, covariances, skewness, and kurtosis from mixture models.	$ obsMeans $ mixedMeans $ mixedMeansResid $ obsCovs $ mixedCovs $ mixedCovsResid $ obsSkewness $ mixedSkewness $ mixedSkewnessResid $ obsKurtosis $ mixedKurtosis $ mixedKurtosisResid	num [1, 1:4] 5.84 3.06 … num [1, 1:4] 5.84 3.06 … num [1, 1:4] 0 0 0 0 … num [1:4, 1:4] 0.681−0.042 … num [1:4, 1:4] 0.681−0.121 … num [1:4, 1:4] 0 0.079 … num [1, 1:4] 0.312 0.316 … num [1, 1:4] −0.037 0.075 … num [1, 1:4] 0.349 0.241 … num [1, 1:4] −0.574 0.181 … num [1, 1:4] −0.615−0.285 … num [1, 1:4] 0.041 0.466 …
$ class_counts	For models that include categorical latent variables (TYPE = MIXTURE), a list of estimated class counts and posterior classification probabilities.	$ modelEstimated	class count proportion 1 497.01 0.25 2 479.66 0.24 3 527.79 0.26
		$ posteriorProb	
		$ mostLikely	
		$ avgProbs.mostLikely	1 2 3 1 0.954 0.012 0.034 2 0.047 0.541 0.412 3 0.051 0.046 0.903
		$ classificationProbs.mostLikely	
		$ logitProbs.mostLikely	
$ lcCondMeans	Equality tests of means across latent classes provided when AUXILIARY variables are included with options *BCH, DCON, E*, or *R3STEP*. Typically includes both omnibus and pairwise comparisons.	$ overall	var m.1 se.1 m.2 se.2 m.3 Z 0.041 0.045−0.084 0.046−0.115 se.3 m.4 se.4 chisq df p 0.044 0.045 0.045 10.371 3 0.016
		$ pairwise	var classA classB chisq df p 1 Z 1 2 3.752 1 0.053 2 Z 1 3 6.152 1 0.013 3 Z 1 4 0.004 1 0.947 4 Z 2 3 0.242 1 0.623 5 Z 2 4 4.001 1 0.045 6 Z 3 4 6.474 1 0.011

(*Continued*)

TABLE 3
(Continued)

Element	Description	Common Subelements	Example of Selected Contents
$ gh5	Contents of hdf5 file typically created by the PLOT command in M*plus*. Contains information used by M*plus* to generate graphs, as well as other useful descriptive statistics.	$ categorical_data $ individual_data $ irt_data $ means_and_variances	$ raw_data: num [1:21, 1:500] 0 0 1 1 ... $ loading : num [1, 1:20, 1] 1.026 ... $ mean : num [1, 1] 0 $ scale : num [1:20, 1] 1 1 1 1 1 ... $ tau : num [1, 1:20, 1] −0.368 ... $ univariate_table: num [1:2, 1:20, 1] 0.426 ... $ variance : num [1, 1] 1 $ estimated_probs:List of 1 ..$ values: num [1:40, 1] 0.425 0.575 ... $ observed_probs:List of 1 ..$ values: num [1:40, 1] 0.426 0.574 ...
$ bparameters	An *mcmc.list* object containing MCMC draws from the posterior distributions from each chain in a Bayesian model. Output by M*plus* when ESTIMATOR = BAYES and SAVEDATA: BPARAMETERS is specified. Separates burn-in draws (prior to chain convergence) from valid draws.	$ burn_in $ valid_draw	$ 1: mcmc [1:500, 1:89] 1 1 1 1 1 1 1 1 1 1 ... ..- attr(*, "dimnames") = List of 2 ....$: chr [1:500] "1" "2" "3" "4" ... .$: chr [1:89] "Chain.number" "MEANDEP" ... ..- attr(*, "mcpar") = num [1:3] 1 500 1 $ 2: mcmc [1:500, 1:89] 2 2 2 2 2 2 2 2 2 2 ... ..- attr(*, "dimnames") = List of 2 ....$: chr [1:500] "24" "25" "26" "27" ... .$: chr [1:89] "Chain.number" "MEANDEP" ... ..- attr(*, "mcpar") = num [1:3] 1 500 1 - attr(*, "class") = chr "mcmc.list"
$ fac_score_stats	Means, covariances, and correlations for latent factor scores (from output section labeled SAMPLE STATISTICS FOR ESTIMATED FACTOR SCORES)	$Means $Covariances $Correlations	F1 F1_SE F2 F2_SE 0 0.486 0 0.478 F1 F1_SE F2 F2_SE F1 0.761 NA NA NA F1_SE 0.000 0.002 NA NA F2 0.484 0.000 0.771 NA F2_SE 0.000 −0.001 0.000 0.001 F1 F1_SE F2 F2_SE F1 0.761 NA NA NA F1_SE 0.000 0.002 NA NA F2 0.484 0.000 0.771 NA F2_SE 0.000−0.001 0.000 0.001

Note. When a single M*plus* output file is specified, *readModels* returns an object of class *mplus.model* that is based on or inherits the *list* data type in R. Thus, each field or subfield within the object is typically accessed using the $ operator. The contents of the *mplus.model* object vary depending on the output. For example, if plot data are requested using the PLOT: command in M*plus*, this will typically result in a .gh5 file, which is read into the $gh5 field of the object. If one specifies a directory as the target of *readModels*, all output files in the directory are read into a single *mplus.model.list* that inherits from the *list* object whose elements are *mplus.model* objects. AIC = Akaike's information criterion; BIC = Bayesian information criterion; RMSEA = root mean square error of approximation; TLI = Tucker–Lewis Index; CFI = comparative fit index.

Simulation conditions varied in terms of sample size, mean indicator separation across latent classes, and the proportion of noise observations added to each replication. For each cell in the simulation design, 1,000 replications were generated using the *MixSim* package in R (Melnykov, Chen, & Maitra, 2012), which supports latent class simulation with noise observations. The levels of sample size were n = {45, 90, 180, 360, 720, 1,440}, where data sets between 50 and 500 participants are common in psychological research (cf. Ning & Finch, 2004). The levels of mean indicator separation across classes were Cohen's d = {.25, .5, 1.0, 1.5, 2.0, 3.0} where indicators were drawn from the unit-normal distribution. Also, indicator means were consistently rank-ordered across latent classes, potentially representing low, medium, and high subtypes of an underlying construct. The proportions of noise observations added to each replication were p = {0, .025, .05, .10, .15, .20, .30}. We defined noise observations as (a) falling outside the 99% multivariate ellipsoids of concentration for all clusters (following Maitra & Ramler, 2009), and (b) uniformly distributed within one-and-a-half times the interquartile range of each indicator. Figure 2 depicts an example of the indicator distributions in the 15% noise condition.

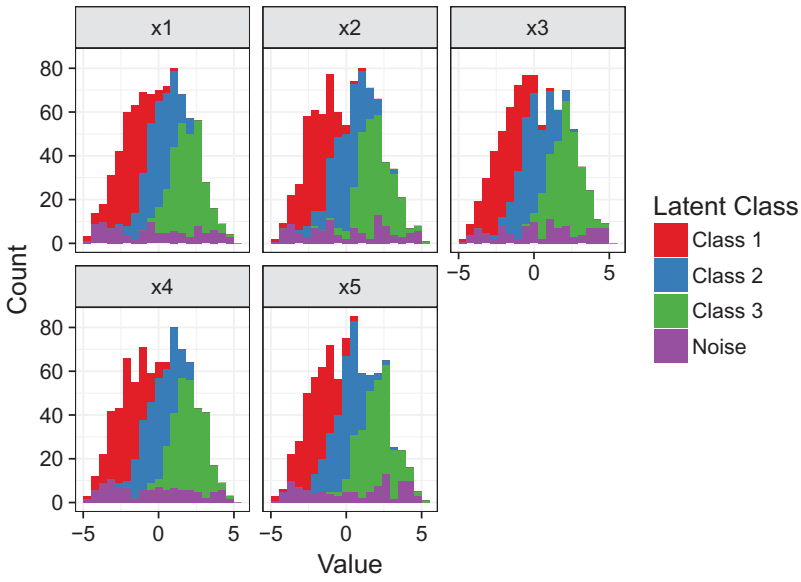


FIGURE 2 Univariate distributions of data simulated from a three-class latent class model (n = 720) including an additional 15% of random noise observations. Colors denote the known latent class membership, as well as the observations drawn from the noise process.

All replications were analyzed in M*plus* version 7.31 using both one-factor CFA (misspecified) and two- to five-class LCA models (where only the three-class model matches the simulation). One could easily extend the simulation and analysis conditions to support additional questions such as how data simulated from a CFA and analyzed under an LCA are affected by noise observations. Altogether, the pipeline for the Monte Carlo study was as follows: (a) simulate 1,000 replications of a multivariate normal Gaussian mixture model for each cell in the simulation design (sample size, indicator separation, and noise proportion); (b) analyze each replication according to CFA and LCA models; (c) extract measures of parameter coverage (especially latent means) and model fit; (d) compare model fit indexes to form a distribution of relative evidence for categorical versus dimensional representations for each replication; and (e) visualize and summarize substantive findings of the study (for a more in-depth tutorial on the steps of a Monte Carlo SEM study, see Paxton, Curran, Bollen, & Kirby, 2001). A secondary feature of this illustration is that it leverages parallel computing facilities in R such that data generation, model estimation, and output extraction can be divided across many processing cores to obtain results much faster than running each replication sequentially.

R code for the data generation step (*GenData_MixSim.R*) is provided in the examples subfolder of the *MplusAutomation* Github repository (https://github.com/michaelhallquist/MplusAutomation) and is not specific to M*plus*. Nevertheless, a general point is that using core programming constructs such as conditional logic and nested loops in R provides a framework for simulating data systematically across a wide range of conditions. The data storage and file system functions of R also provide tools to save replications in a highly compressed format and to organize replication files in a comprehensible subfolder structure for storage and management. Here, we have organized the 1,000 replications for each condition into a single .RData file, with subfolders named by the sample size, latent mean separation, and level of noise. For example, all replications for the n = 45, p = 0, d = 1 condition are located in `mplus_scatter/n45/p0/s1/c3_v5.RData` where c3 denotes a three-class LCA and v5 denotes five indicator variables. After simulating the data, the script generates a queue of models to be run in M*plus*, consisting of one-factor CFA and two- to five-class LCA models for each cell in the simulation design matrix.

Two scripts (*RunFMM_MCReps.R* and *RunCFA_MCReps.R*) then loop through the queues and estimate the model corresponding to a given simulation condition in M*plus*. To speed up this process substantially, the `runModels function` is executed in parallel using the *doSNOW* and *foreach* packages (Calaway, Revolution Analytics, & Weston, 2015a, 2015b). Because model estimation for each replication does not depend on the details of any other model (i.e., there is no communication among estimation jobs), running in parallel leads to a nearly linear speed-up such that having twice as many cores halves the computation time for the queue. Only the number of available cores and compliance with the M*plus* license agreement limit this speed-up. To take advantage of parallel processing without cluttering the file system with thousands of data, input, and output files for M*plus*, the estimation scripts create a unique temporary directory for each model

estimation, then remove the files after models have been read into R using the `readModels` function. This strategy also prevents potential output file collision problems among M*plus* jobs running in parallel. Altogether, the core estimation process for each condition is (a) load the simulated replications from a compressed .RData file into memory; (b) create a temporary directory; (c) write all replications to plain text files compatible with M*plus*; (d) generate an M*plus* input syntax file corresponding to the condition; (e) estimate the model for each replication using `runModels`; (f) read the model output using `readModels`, storing the results into a list of mplus.model objects; and (g) clean up the temporary files and save the results list to a file for further analysis. See Appendix D for a code-based synopsis of the analytic pipeline. Full code for the generation and estimation of all models is located in the examples folder of the *MplusAutomation* code repository at https://github.com/michaelhallquist/MplusAutomation.

For this project, we could have used an external Monte Carlo approach in M*plus* to obtain estimates of parameter coverage and mean squared error, as well as average fit statistics, across replications (e.g., see M*plus* User's Guide Example 12.6; Muthén & Muthén, 2017). To illustrate the extensibility and benefits of *MplusAutomation*, however, we chose to run each replication in each condition separately, storing the all model outputs extracted by readModels. The major advantage of this approach is that any model output supported by the package is available for extraction and secondary analysis, rather than being limited to the summary statistics provided by M*plus* (e.g., 95% parameter coverage). For example, if one were interested in characterizing the moments of posterior distributions across replications in a Bayesian modeling context, one could include SAVEDATA: BPARAMETERS in the M*plus* syntax and access the $bparameters field of the corresponding `mplus.model` object containing draws from the posterior for each replication. Likewise, in a mixture modeling context, one could count how many of the replications the BLRT supported a *k*-class model compared to the $k - 1$-class model.

More specific to our example, extracting output for each replication allows us to generate a relative evidence distribution comparing CFA and LCA models. Model comparison statistics based on the log-likelihood will differ somewhat across replications due to the sampling variability inherent in the simulation process. However, relative differences in model selection criteria within a replication are ratio-distributed and interpretable in terms of Bayes factors (for BIC) or Akaike weights (for AIC; Burnham & Anderson, 2002). Thus, to avoid the arbitrary likelihood scaling differences across replications in deriving a distribution of relative evidence, we computed the difference in AIC and BIC for the CFA and LCA models of each replication.

Although there are many potential results of this *MplusAutomation* Monte Carlo example, we focus on the relative difference in evidence for CFA versus three-class LCA models as a function of information criterion (AIC, BIC, and corrected AIC), latent mean separation, and noise. As depicted in Figure 3, at a sample size of 180, the BIC is prone to selecting a one-factor CFA over a three-class LCA when latent mean separation among indicators is small to moderate ($0.25 \leq d \leq 1.0$) and the level of noise is low ($\leq 15\%$), whereas the corrected AIC prefers the data-generating three-class LCA under most of the simulation conditions, perhaps speaking to differences in the asymptotic properties of these criteria (consistency vs. efficiency; Yang, 2005). Another interesting result is that increasing the level of noise shifts model selection toward the three-class LCA for latent mean separations of $d \leq 1.0$, but degrades relative evidence of the LCA over CFA at higher latent mean separation. In a substantive study, such preliminary findings might suggest further analyses of how noise observations are assigned to latent classes and whether the indicator means are recovered accurately in each class despite noise. Altogether, the *MplusAutomation* package facilitates Monte Carlo studies in *M*plus by allowing users to run many simulation conditions unattended and in parallel, as demonstrated by the 252,000 separate models estimated in the current example. The package also allows for detailed statistics of each replication to be stored in R data structures to support detailed analyses of specific parameters and statistics.

OTHER FEATURES OF *MPLUSAUTOMATION*

Although the *MplusAutomation* package has been developed primarily by Hallquist and Wiley, we are grateful to a broader community of users for contributing additional features to the package and for providing output files to improve compatibility with new versions of M*plus*. Here, we highlight a few specific features that extend the capabilities of M*plus*. First, for Bayesian SEM, `readModels` extracts draws from the joint posterior distribution into structures compatible with the *coda* package (Plummer, Best, Cowles, & Vines, 2006), which has diagnostic functions for Markov chain Monte Carlo (MCMC) estimation. These draws are stored in the `$bparameters` field when Bayesian models include the SAVEDATA: BPARAMETERS command in M*plus*. Extending such functionality, the `mplus.traceplot` function plots MCMC estimates from successive draws of a chain, which is a common diagnostic check for chain convergence and stationarity. Related, users can conduct informative hypothesis tests in a Bayesian framework by comparing the number of draws from the posterior that corroborate versus refute a given hypothesis, which can be a function of a single condition (`testBParamConstra-int`) or multiple conditions (`testBParamCompound-Constraint`) (Van De Schoot, Hoijtink, Hallquist, & Boelen, 2012).

Identifying the source of estimation problems in SEM (e.g., negative variance estimates) is often crucial to identifying misspecification or achieving proper fit. M*plus* provides a

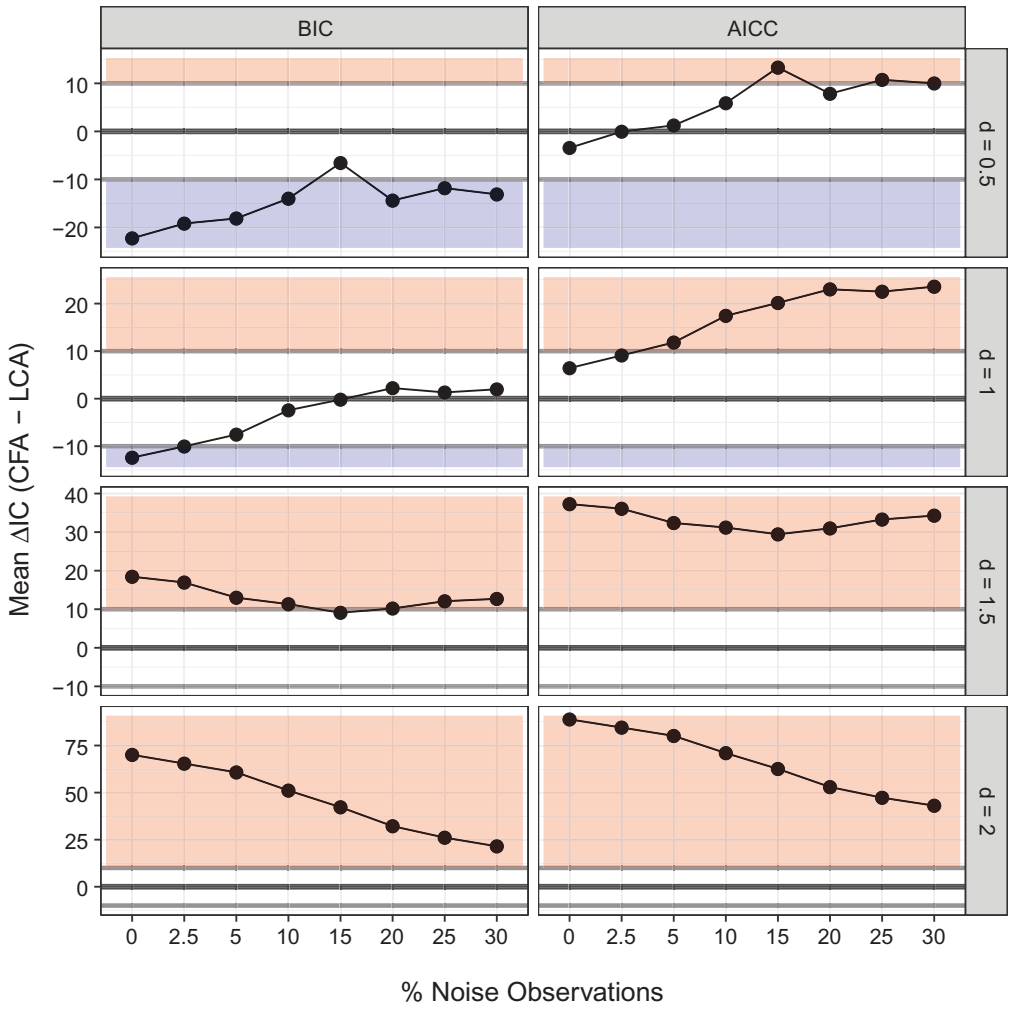


FIGURE 3 The difference in relative model evidence as function of random noise level and latent separation among indicators. Mean relative model evidence (plotted on the *y* axis) is measured by the difference in an information criterion (IC) for a confirmatory factor analysis (CFA) versus three-class latent class analysis (LCA), where differences of 10 points or more are considered strong evidence of one model over another (Burnham & Anderson, 2002). Positive values of the IC differences indicate greater support for the LCA model over the CFA (light red rectangles), whereas negative values denote the converse (light blue rectangles). Data were generated from a three-class LCA, so positive IC difference values represent relative evidence of the data-generating model over a dimensional model. Each dot on the plot represents the mean IC difference across 1,000 replications for that simulation condition. BIC = Bayesian information criterion; AICC = corrected Akaike's information criterion.

useful text description in the warning and errors section of model output that is read into the `$warnings` and `$errors` elements of an `mplus.model` object by `readModels`. Users can extract warnings and errors from a large set of models stored as an `mplus.model.list` using conventional R syntax for subsetting lists such as `all_warnings <- lapply(mylist, "[[", "warnings")`. Often, warnings and errors name a particular parameter number as problematic, where the numbering of free parameters in matrix form is provided by M*plus* using `OUTPUT: TECH1`, which is read into `$tech1` by `readModels`. To aid in identifying particular parameters for diagnosing model estimation problems, *MplusAutomation* provides the `lookupTech1Parameter` function:

```
> model <- readModels("mymodel.out")
> lookupTech1Parameter(model, 20)
## Matrix name: beta
##
## row col
## C_ACON2 ALC
```

Here, the output indicates that parameter 20 is the regression of `C_ACON2` on ALC in the beta matrix (which contains regressions among continuous latent variables).

In latent growth models, researchers often wish to compare among residual covariance structures to capture the

temporal structure of the data (e.g., first-order autoregressive; Bollen & Curran, 2005). *MplusAutomation* provides the mplusRcov function to facilitate testing temporal covariance structures. Data must be in wide format (i.e., a separate variable for each time point). The mplusRcov function requires two arguments: (a) the variable names, ordered by time; and (b) the type of residual structure to test. For example, for a homogenous residual structure, mplusRcov generates M*plus* model syntax to constrain the variances to equality and fix all covariances at zero. For a compound symmetric structure, it generates syntax to constrain variances to equality and constrain all covariances to equality. When necessary, `mplusRcov` also produces syntax for the MODEL CONSTRAINT section of M*plus* (e.g., for autoregressive structures). The syntax from mplusRcov can either be pasted into an input file or integrated directly in R using the update function and the `mplusModeler` framework described earlier.

LIMITATIONS OF *MPLUSAUTOMATION*

One of the most attractive features of M*plus* is its support for many models, diagnostics, and fit statistics. Because M*plus* output is largely provided in text form, a challenge of such extensibility is that we must develop code to parse each section. As a result, the package does not yet support extraction of some sections such as factor loadings from exploratory factor analysis models. Finally, *MplusAutomation* does not have access to the internals of M*plus*; only information provided in output files is available for extraction into R. As a result, we encourage users who wish to implement new methods that affect model parameterization or estimation to work directly with the developers of M*plus*.

CONCLUSION

The *MplusAutomation* package provides an interface between M*plus* and R to automate and streamline the creation, estimation, and interpretation of latent variable analyses. The current version of *MplusAutomation* (v0.7 as of this writing) supports outputs from M*plus* version 8, released in April 2017, which provides time series analyses and other methodological innovations. Although the package does not extract all aspects of M*plus* output, the code base has been developed over several years and core functions have been tested extensively. Our hope is that it encourages users to take advantage of the numerous data science strengths of R (Wickham & Grolemund, 2017) to compare, interpret, and report latent variable models. By providing programmatic access to the output of M*plus*, *MplusAutomation* supports reproducible research practices (Gandrud, 2015) and might reduce scientific errors caused by human mistakes (e.g., copy and paste errors). Finally, by reducing the effort required for large-scale SEM studies, a broad goal of *MplusAutomation* is to support methodological developments using Monte Carlo studies that depend on the management of many syntax files.

FUNDING

This research was supported by grants from the National Institute of Mental Health to M.N.H. (F32 MH090629, K01 MH097091).

ORCID

Michael N. Hallquist http://orcid.org/0000-0001-5894-8038

REFERENCES

Arbuckle, J. L. (2014). Amos (Version 23.0) [Computer software]. Chicago, IL: IBM SPSS.

Asparouhov, T., & Muthén, B. (2014). Auxiliary variables in mixture modeling: Three-step approaches using M*plus*. *Structural Equation Modeling*, *21*, 329–341. doi:10.1080/10705511.2014.915181

Bollen, K. A., & Curran, P. J. (2005). *Latent curve models: A structural equation perspective*. New York, NY: Wiley-Interscience.

Burnham, K. P., & Anderson, D. R. (2002). *Model selection and multimodel inference: A practical information-theoretic approach* (2nd ed.). New York, NY: Springer.

Calaway, R., & Weston, S. (2015a). *doSNOW: Foreach parallel adaptor for the "snow" package*. Retrieved from https://CRAN.R-project.org/package=doSNOW

Calaway, R., & Weston, S. (2015b *foreach: Provides foreach looping construct for R*. Retrieved from https://CRAN.R-project.org/package=foreach

Enders, C. K., & Bandalos, D. L. (2001). The relative performance of full information maximum likelihood estimation for missing data in structural equation models. *Structural Equation Modeling*, *8*, 430–457. doi:10.1207/S15328007SEM0803_5

Gandrud, C. (2015). *Reproducible research with R and R Studio* (2nd ed.). Boca Raton, FL: Chapman & Hall/CRC.

Hallquist, M. N., & Pilkonis, P. A. (2012). Refining the phenotype for borderline personality disorder: Diagnostic criteria and beyond. *Personality Disorders: Theory, Research, and Treatment*, *3*, 228–246. doi:10.1037/a0027953

Hallquist, M. N., & Wright, A. G. C. (2014). Mixture modeling in the assessment of normal and abnormal personality: I. Cross-sectional models. *Journal of Personality Assessment*, *96*, 256–268. doi:10.1080/00223891.2013.845201

Hox, J., van de Schoot, R., & Matthijsse, S. (2012). How few countries will do? Comparative survey analysis from a Bayesian perspective. *Survey Research Methods*, *6*, 87–93.

Jöreskog, K. G., & Sörbom, D. (2015). LISREL 9.20 for Windows [Computer software]. Skokie, IL: Scientific Software International.

Kline, R. B. (2015). *Principles and practice of structural equation modeling* (4th ed.). New York, NY: Guilford.

Maitra, R., & Ramler, I. P. (2009). Clustering in the presence of scatter. *Biometrics*, *65*, 341–352. doi:10.1111/j.1541-0420.2008.01064.x

Markon, K. E., & Krueger, R. F. (2006). Information-theoretic latent distribution modeling: Distinguishing discrete and continuous latent variable models. *Psychological Methods*, *11*, 228–243. doi:10.1037/1082-989X.11.3.228

Melnykov, V., Chen, W.-C., & Maitra, R. (2012). MixSim: An R package for simulating data to study performance of clustering algorithms. *Journal of Statistical Software*, *51*(12), 1–25. doi:10.18637/jss.v051.i12

Merkle, E. C., You, D., & Preacher, K. J. (2016). Testing nonnested structural equation models. *Psychological Methods*, *21*, 151–163. doi:10.1037/met0000038

Muthén, B., & Asparouhov, T. (2012). Bayesian structural equation modeling: A more flexible representation of substantive theory. *Psychological Methods*, *17*, 313–335. doi:10.1037/a0026802

Muthén, L. K., & Muthén, B. O. (2002). How to use a Monte Carlo study to decide on sample size and determine power. *Structural Equation Modeling*, *9*, 599–620. doi:10.1207/S15328007SEM0904_8

Muthén, L. K., & Muthén, B. O. (2017). *M*plus *user's guide* (version 8). Los Angeles, CA: Muthén & Muthén.

Neale, M. C., Hunter, M. D., Pritikin, J. N., Zahery, M., Brick, T. R., Kickpatrick, R. M., … Boker, S. M. (2016). OpenMx 2.0: Extended structural equation and statistical modeling. *Psychometrika*, *81*, 535–549. doi:10.1007/s11336-014-9435-8

Ning, Y., & Finch, S. J. (2004). The likelihood ratio test with the box-cox transformation for the normal mixture problem: Power and sample size study. *Communications in Statistics: Simulation and Computation*, *33*, 553–565. doi:10.1081/SAC-200033328

Nylund, K. L., Asparouhov, T., & Muthén, B. O. (2007). Deciding on the number of classes in latent class analysis and growth mixture modeling: A Monte Carlo simulation study. *Structural Equation Modeling*, *14*, 535–569. doi:10.1080/10705510701575396

Paxton, P., Curran, P. J., Bollen, K. A., & Kirby, J. (2001). Monte Carlo experiments: Design and implementation. *Structural Equation Modeling*, *8*, 287–312. doi:10.1207/S15328007SEM0802_7

Plummer, M., Best, N., Cowles, K., & Vines, K. (2006). CODA: Convergence diagnosis and output analysis for MCMC. *R News*, *6*(1), 7–11.

R Core Team. (2015). *R: A language and environment for statistical computing* (Version 3.2.3). Vienna, Austria: R Foundation for Statistical Computing.

Rosseel, Y. (2012). lavaan: An R package for structural equation modeling. *Journal of Statistical Software*, *48*(2), 1–36. doi:10.18637/jss.v048.i02

van de Schoot, R., Hoijtink, H., Hallquist, M. N., & Boelen, P. A. (2012). Bayesian evaluation of inequality-constrained hypotheses in SEM models using M*plus*. *Structural Equation Modeling*, *19*, 593–609. doi:10.1080/10705511.2012.713267

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis* (2nd ed.). Cham, Switzerland: Springer International. doi:10.1007/978-3-319-24277-4

Wickham, H., & Grolemund, G. (2017). *R for data science: Import, tidy, transform, visualize, and model data*. Sebastopol, CA: O'Reilly Media.

Yang, Y. (2005). Can the strengths of AIC and BIC be shared? A conflict between model identification and regression estimation. *Biometrika*, *92*, 937–950. doi:10.1093/biomet/92.4.937

APPENDIX A
Example Template File for Generating M*plus* Input Files for Latent Class and Confirmatory Factor Models

```
[[init]]
iterators = model;
model=1:5;
modelnames#model = CFA LCA2 LCA3 LCA4 LCA5;
title#model = "Confirmatory Factor Analysis"
                  "2-class LCA"
                  "3-class LCA"
                  "4-class LCA"
                  "5-class LCA";

classes#model = 0 2 3 4 5;

filename = "[[modelnames#model]].inp";
outputDirectory = templateExample/
  [[modelnames#model]];
[[/init]]

TITLE: [[title#model]]
DATA: FILE = "../../lca_cfa_example.dat";
VARIABLE: NAMES = u1-u6; MISSING=.;
         CATEGORICAL = u1-u6;

[[model > 1]] CLASSES = c([[classes#model]]);
  [[/model > 1]]

ANALYSIS:
[[model == 1]] TYPE=GENERAL; ESTIMATOR=MLR;
  [[/model == 1]]
[[model > 1]] TYPE = MIXTURE; STARTS = 250 25;
  [[/model > 1]]

MODEL:
[[model == 1]] factor BY u1-u6; [[/model == 1]]
[[model > 1]] %OVERALL% [[/model > 1]]

OUTPUT: TECH1 TECH8 [[model > 1]] TECH14
  [[/model > 1]];
PLOT: TYPE=PLOT3;
```

APPENDIX B
Generating a Plot of M*plus* Model-Estimated Item Frequencies From a Latent Class Analysis Using Output Extracted by the *readModels* Function

```
library(MplusAutomation); library(reshape2)
library(ggplot2); library(scales); library
  (cowplot)

lcacfa <- readModels("templateExample",
  recursive = TRUE)

p <- as.data.frame(lcacfa[[2]][["gh5"]]
  [["means_and_variances_data"]]
  [["estimated_probs"]][["values"]][seq(2, 12,
  2),])
colnames(p) <- paste0("Class ", 1:2)
p <- cbind(Var = paste0("U", 1:6), p)
pl <- melt(p, id.vars = "Var")

ggplot(pl, aes(as.integer(Var), value,
  shape = variable, colour = variable)) +
geom_point(size = 3) +
scale_x_continuous("", breaks = 1:6, labels = p
  $Var) +
scale_y_continuous("Percent of 1s (vs 0s)", labels
  = percent) +
scale_colour_manual(values = c("Class 1"="black",
  "Class 2"="grey50")) +theme_cowplot() +
theme(legend.key.width = unit(1.5, "cm"),
   legend.title = element_blank(),
   legend.position = "bottom") +
coord_cartesian(xlim = c(.9, 6.1), ylim = c(0, 1),
  expand = FALSE)
```

Note. Lines 1 and 2 load several R packages for data management and graphing. The code for the graph (Lines 12–22) is the most complex, although a basic graph could be generated with just Lines 12 to 14. The remaining lines customize the color, shapes, labels, and axis range of the graph.

APPENDIX C
Detailed Comparison of Two Confirmatory Factor Analysis Models Using *compareModels* When Adding a Residual Covariance Between Two Items (PAR4 and BORD4)

```
> faModels <- readModels("CFA")
> compareModels(faModels[["dsm3_top8_CFA_MLR_MI_ParBord.out"]],
   faModels[["dsm3_top8_CFA_MLR.out"]], diffTest=TRUE, equalityMargin=.01,
   show=c("summaries", "diff", "unique"))
==============
Mplus model comparison
----------------------
Model 1: ./dsm3_top8_CFA_MI_ParBord.out
Model 2: ./dsm3_top8_CFA.out

Model Summary Comparison
------------------------
                    m1                          m2
Title               Basic CFA of top 8          Basic CFA of top 8, allowing par4-bord4
                                                  resid cov
Observations        303                         303
Estimator           WLSM                        WLSM
Parameters          25                          24
ChiSqM_Value        49.522                      82.656
ChiSqM_DF           19                          20
CFI                 0.98                        0.959
TLI                 0.971                       0.943
RMSEA_Estimate      0.073                       0.102
WRMR                0.815                       1.068
   WLSM Chi-Square Difference test for nested models
   -------------------------------------------------
   Difference Test Scaling Correction: 0.94
   Chi-square difference: 22.3357
   Diff degrees of freedom: 1
   P-value: 0


   Note. The chi-square difference test assumes that these models are nested. It is up to you to verify this
assumption.
   =========
   Model parameter comparison
   --------------------------
   Parameter estimates that differ between models (param. est. diff > 0.01)
   ------------------------------------
   paramHeader  param   m1_est  m2_est  . m1_se  m2_se  .  m1_est_se  m2_est_se  .  m1_pval  m2_pval
        GPD.BY  BORD4    0.649   0.767  | 0.076 0.070  |      8.577     10.891  |        0         0
        GPD.BY   PAR4    0.667   0.769  | 0.072 0.067  |      9.286     11.449  |        0         0
   Variances      GPD    0.735   0.707  | 0.068 0.064  |     10.806     11.009  |        0         0
   Parameters unique to model 1: 1
   -------------------------------
   paramHeader param m1_est m1_se m1_est_se m1_pval
     PAR4.WITH BORD4  0.288  0.057     5.033        0
   Parameters unique to model 2: 0
   -------------------------------
   None
```

APPENDIX D
R Code Synopsis of Parallel Execution of Model Replications for All Latent Class Models

```
replications <- 1:1000
load("lcaMasterList.RData") #contains lcalist

library(doSNOW); library(foreach)
njobs <- 50; clusterobj <- makeSOCKcluster(njobs); registerDoSNOW(clusterobj)

fmm <- foreach(m = iter(lcalist), .inorder = FALSE, .multicombine = TRUE,
       .packages = c("MplusAutomation")) %dopar% {

repResults <- list()
for (i in 1:min(length(fmm_sim), max(replications))) {
   #write syntax for this replication to a temporary directory
   cat(getSyntax(m$fc, m$nc, m$nv, m$ss, m$sep, m$np, dataSection = dataSection,
        propMeans = propMeans, means = means, output = outputSection,
        savedata = savedata), file = file.path(tempdir(), "individual_rep.inp"))

  runModels(tempdir(), recursive = FALSE, logFile = NULL)
  repResults[[i]] <- readModels(file.path(tempdir(), "individual_rep.out"))
}

cleanupTempFiles(m) #remove various files used by Mplus
return(repResults)
}

stopCluster(clusterobj)
```

Note. Full R code for all steps in the pipeline is located in the *MplusAutomation* Github repository at https://github.com/michaelhallquist/MplusAutomation/tree/master/examples/monte_carlo.