Implementing a Simulation Study Using Multiple Software Packages for Structural Equation Modeling

Sunbok Lee

Abstract
A Monte Carlo simulation study is an essential tool for evaluating the behavior of various quantitative methods including structural equation modeling (SEM) under various conditions. Typically, a large number of replications are recommended for a Monte Carlo simulation study, and therefore automating a Monte Carlo simulation study is important to get the desired number of replications for a simulation study. This article is intended to provide concrete examples for automating a Monte Carlo simulation study using some standard software packages for SEM: Mplus, LISREL, SAS PROC CALIS, and R package lavaan. Also, the equivalence between the multilevel SEM and hierarchical linear modeling (HLM) is discussed, and relevant examples are provided. It is hoped that the codes in this article can provide some building blocks for researchers to write their own code to automate simulation procedures.

Keywords
Monte Carlo simulation, automation.

Introduction
Evaluating the behavior of estimators under various conditions is important to ensure the validity of inferences based on structural equation modeling (SEM). For example, the violation of the multivariate normality assumption (Finney & DiStefano, 2006) and the effect of missing data (Enders & Bandalos, 2001) could be investigated by evaluating the behavior of estimators under those specific conditions. Analytic solutions to those questions could be obtained under some ideal assumptions such as large sample sizes in asymptotic theory. However, analytic solutions are not always tractable especially when the conditions of interest become more realistic (small sample size, non-normality, etc.). A Monte Carlo simulation study is a useful alternative for evaluating the behavior of estimators when analytic solutions are not available.

A Monte Carlo simulation study is a computer-intensive procedure in which random numbers are used to get empirical sampling distributions of estimators under conditions of interest (Bandalos, 2006). In a Monte Carlo simulation study, a large number of samples are replicated based on known population parameter values, and empirical sampling distributions are obtained from the set of estimates calculated across replicated samples. The empirical sampling distributions can be used to evaluate the behavior of estimators in terms of bias, root mean square error, coverage rate, and so on.

The procedures of a Monte Carlo simulation study can be conceptualized into design and implementation stages (Paxton, Curran, Bollen, Kirby, & Chen, 2001). In a design stage, simulation conditions such as the number of replications, distributions of variables, and methods for estimation are chosen based on research questions. Given simulation conditions, a Monte Carlo simulation study is implemented using a specific software package to estimate parameters of the model of interest. A large number of replications are typically recommended for a Monte Carlo simulation study, and therefore automating a Monte Carlo simulation study is essential to get the desired number of replications for a simulation study.

This article was intended to provide concrete examples of automating a Monte Carlo simulation study using some standard software packages for SEM. To do so, the capabilities of a Monte Carlo simulation study in various SEM software packages were demonstrated using the example of a simple confirmatory factor analysis (CFA) with two factors. Also, as an extension of the standard SEM, a multilevel SEM was compared with hierarchical linear modeling (HLM), and relevant examples were also provided. More specifically, this work is guided by three goals: (a) to review built-in Monte Carlo simulation capabilities in Mplus (L. K. Muthén &
Monte Carlo Simulation Studies in SEM

To do Monte Carlo simulation studies, several factors need to be considered in the design and implementation stages. For example, the number of replications in a Monte Carlo simulation study, which corresponds to the sample size in applied research, is directly related to the sampling variance of estimated parameters. Hence, the number of replications needs to be chosen based on the accuracy required by the purpose of the simulation study (Cohen, Kane, & Kim, 2001; Harwell, Stone, Hsu, & Kirisci, 1996). Also, the values of population parameters need to be chosen to reflect conditions commonly encountered in applied research studies (Paxton et al., 2001). The guidelines for a Monte Carlo simulation study in general are well documented in previous literature (Bandalos, 2006; Harwell et al., 1996). In this article, the focus is given on how to implement or automate a Monte Carlo simulation study using standard software packages for SEM.

Various traditional software packages are available for the estimation in SEM. Comparisons of such software packages (Narayanan, 2012; West & Galecki, 2011) are useful for researchers to choose software packages appropriate for their analytic goals. In this article, four software packages—Mplus 7 (L. K. Muthén & Muthén, 1998-2012), LISREL 8.7 (Jöreskog & Sörbom, 2006), R package lavaan (Rosseel, 2012), and PROC CALIS in SAS 9.3 (SAS Institute, Cary, NC, USA)—were chosen to demonstrate actual implementations of a Monte Carlo simulation study in SEM. Also, SEM is flexible enough to model multilevel data structure, and it is well known that multilevel SEM is equivalent to HLM (Hox, 2013; Mehta & Neale, 2005). In this article, the R code that can automate a Monte Carlo simulation study using HLMwin was also provided and explained, which could be useful in conducting a Monte Carlo simulation study comparing multilevel SEM and HLM. The relevant R functions that were used to automate a Monte Carlo simulation study using HLMwin could be useful in automating a Monte Carlo simulation study using other software packages that do not have built-in Monte Carlo capabilities.

For Mplus, LISREL, and PROC CALIS, built-in capabilities for Monte Carlo simulations are reviewed, with the emphasis on doing simulations using data sets that are generated outside those software packages. Often, researchers prefer to use their own code in generating data sets to precisely control the data generation mechanism. In Mplus, LISREL, and PROC CALIS, Monte Carlo simulation studies can be performed using external data sets that are generated by researchers’ own codes as well as the data sets that are generated by those software packages themselves. On the other hand, R package simsem is used to facilitate a simulation study using the R package lavaan. Moreover, the R package simsem can also be used to generate data sets for a simulation study using Mplus and LISREL. For HLMwin, the R code that automates simulation procedures for writing input command files, running HLMwin, and extracting statistics from output files is provided.

A Simulation Study Using Mplus

Mplus. Mplus provides comprehensive capabilities for a Monte Carlo simulation study. One way of doing a simulation study using Mplus is to analyze data sets that are generated by Mplus itself. In Mplus, data sets for a simulation can be generated based on the population parameters specified in the MODEL POPULATION command and analyzed based on the model specified in the MODEL command. In the MODEL POPULATION command, population parameters can be fixed to a specific number or freed to be estimated using the @ or * symbol, respectively, followed by numbers representing the values of population parameters. The results of simulations are presented using summary statistics such as the average and standard deviation of parameter estimates, average of the estimated standard errors, mean square error for each parameter, and coverage rate of 95% confidence intervals.

Often, researchers prefer to generate data sets using their own code to control the details of the data generation mechanism. In Mplus, external data files can be used to perform a Monte Carlo simulation study. To use external data files, the text file specified in the FILE option of the DATA command needs to contain a list of file names of the external data files and the TYPE = MONTECARLO option should be used in the DATA command.

On the other hand, Mplus can be used as a data generator for Monte Carlo simulation studies using other software packages. Using the REPSAVE and SAVE options in the MONTECARLO command, the data sets generated by the MODEL POPULATION command can be saved as text files. For example, REPSAVE = ALL and SAVE = sim*.dat options can be used to save data files for all replications with the file names, sim1.dat, sim2.dat, and so on. The list of data...
file names are automatically saved in the file named simlist.dat. Those generated data files can be used for a Monte Carlo simulation study using other software packages.

**Example codes for a simulation using Mplus.** In the Mplus codes in Appendix A1, data sets for a simulation are generated using Mplus itself. A simple CFA with two factors is specified in the MODEL POPULATION command. The factor loadings are set to be freely estimated using * symbol and the variances are set to be fixed to specific values using the @ symbol. The values of parameters in the MODEL command are used as starting values in estimating parameters or as population parameters in calculating coverage rates. By using the REPSAVE and SAVE options in the MONTECARLO command, data sets for all the replications are saved as text files named sim1.dat, sim2.dat, and so on.

In the Mplus code in Appendix A2, it is assumed that data sets for a simulation are generated outside Mplus. For this example, data files are created using the R package simsem as presented in Appendix C. In the simsem package, a user can specify a model for data generation and the generated data sets can be exported to text files using the exportData() function. The current version of the simsem package supports exporting data files for both Mplus and LISREL. Because the character string for the fileStem option in the exportData() function is specified as “simMplus,” the names of data files created by the exportData() function are simMplus1.dat, simMplus2.dat, and so on, and the file named simMplus.dat contains a list of data file names. In Mplus, to do a Monte Carlo simulation using external data files, a file that contains a list of external data file names needs to be specified in the FILE option of the DATA command. By running the Mplus code in Appendix A2, Mplus fits the model specified in the MODEL command to data files listed in the data.dat and provides summary statistics for the simulation results.

**A Simulation Study Using R Package lavaan**

The R package lavaan, which stands for a latent variable analysis, is developed for a latent variable modeling in R. This package is still under development, adding new features. For a simulation study using lavaan, researchers can write their own R code from scratch. In that way, researchers can have more control over simulation procedures, especially for the data generation part. The structure of a code for a simulation using lavaan would be similar to the one using HLM in the previous section in that both codes need to generate data sets, repeatedly run software packages to estimate parameters for each replication, and summarize results across replications. However, the whole simulation procedures can be more easily connected in the case of lavaan because all the intermediate results such as generated data sets or estimated parameters are saved as variables in R. In that case, connecting simulation procedures is just a matter of referring variables within R. In contrast, in the case of HLM, additional efforts need to be given to writing input command files or extracting statistics from output text files.

In this article, instead of writing R code from scratch, the R package simsem is used to do a simulation using lavaan. Some built-in functions in the package simsem can be used to generate data sets for a simulation, to run lavaan to estimate parameters, and to summarize estimated parameters across replications. By using those built-in functions, R code for a simulation study using lavaan can be greatly simplified compared with the one written from scratch.

**Some built-in functions in the package simsem.** In the R package simsem, three built-in functions are essential to implement a simulation study using lavaan: the bind(), model(), and sim() function. First, the bind() function creates simMatrix objects that represent various matrices in SEM using LISREL-style notation. For example, LY, PS, and TE symbols represent factor loading matrix, residual variance–covariance matrix among endogenous factors, and measurement error variance–covariance matrix among indicators, respectively. A simMatrix object contains information on the parameter specifications of a model such as freed, fixed, or constrained parameters: Any parameter specified by the string NA is a free parameter to be estimated and the parameters represented by the same character are constrained to be equal. The simMatrix objects created by the bind() function are combined to build templates, which are simSem objects, for data generation and analysis using the model() function. Given the templates, an actual Monte Carlo simulation study can be performed using the sim() function that can generate data sets, analyze data sets using lavaan, and summarize results across replications at the same time. The implementation of a Monte Carlo simulation study using lavaan can be greatly simplified by using the bind(), model(), and sim() function in the R package simsem.

Like Mplus, the R package simsem can also be used as a data generator for a simulation study using other software packages. The data sets generated in the package simsem can be exported to text files using the exportData() function. In the current version, the exportData() function exports data sets to text files that can be analyzed by either Mplus or LISREL.

**Example codes for a simulation using the package simsem.** R codes for a simulation using the package simsem are presented in Appendix C. A simple CFA model with two factors was used for this example. First, to represent six indicators measuring two factors, a six-by-two matrix named “loading” was created to define a matrix for factor loadings. As each factor was measured by three indicators, three NAs were assigned to each column to indicate free parameters to be estimated. The values of population factor loadings were defined using a separate matrix named “loadingValues.” The bind() function was used to create simMatrix object LY using
those matrices. The simMatrix objects PS and TE, which represent residual variance–covariance matrix among endogenous factors, and measurement error variance–covariance matrix among indicators, respectively, were also created in a similar way. Those simMatrix objects are passed to the function model() to create templates, which is a simSem object, for data generation and analysis. Given the simSem object, a Monte Carlo simulation can be easily done using the function sim(). Also, data files for a simulation using other software packages can be created using the function exportData()

A Simulation Study Using LISREL

Like Mplus, external data files can be used to perform a Monte Carlo simulation using LISREL. For a simulation using external data files, LISREL can be run in batch mode using a DOS batch file as shown in Gagne and Furlow (2009) or using the shell() function in R as shown in the simulation study for HLM in the previous section. In this article, the built-in RP command in LISREL is used to do a simulation study using LISREL. With the RP command, LISREL fits a model to multiple data sets in a single data file and reports multiple results in a single output file. The RP command is used to specify the number of multiple data sets or replications within a single data file. With PV option in OU command, all estimated parameters will be saved in a text file. For data generation, researchers’ own code or some software packages such as Mplus or R package simsem can be used. As mentioned in the previous section, the R package simsem is capable of exporting data sets that can be directly analyzed by LISREL using the RP command. In this article, the data sets for a simulation using LISREL in Appendix D are created using the R package simsem in Appendix C.

A Simulation Study Using SAS PROC CALIS

PROC CALIS is a procedure for doing SEM in SAS. As SAS provides programming capabilities, whole simulation procedures, including generating data sets, running PROC CALIS, and summarizing results, can be implemented within SAS as shown in Fan and Fan (2005). In their work, a do-loop statement was used to run PROC CALIS repeatedly. In this article, the BY and ODS (Output Delivery System) statement in PROC CALIS are used to do a Monte Carlo simulation without using a do-loop statement, which makes the code more simple. With the BY statement, PROC CALIS performs separate analyses on observations in groups defined by the variables in the BY statement. Therefore, using the BY statement, analyses across all replications can be done at one time without using a do-loop statement when data sets for all replications are combined into one data set including a group variable representing replications. On the other hand, the ODS output statement can be used to create SAS data sets from internal ODS tables containing parameter estimates, fit statistics, and so on. The names of available ODS tables can be checked in the PROC CALIS manual. Combined with the BY statement, the ODS statement creates SAS data sets that contain results of PROC CALIS across all replications.

Example code for a simulation using PROC CALIS. Example codes for a simulation using PROC CALIS are presented in Appendix E. In this example, data files for a simulation are generated by Mplus using the code presented in Appendix A1. As Mplus creates a separate data file for each replication, the R code in Appendix E1 is used to combine data files into one data file, including a group variable representing replications. Using the SAS code in Appendix E2, the combined data file is imported into SAS, sorted by the variable representing replications, and fitted to a model. With the BY statement, a separate analysis is done for each replication without using a do-loop statement. The parameter estimates across all replications are saved into the SAS data set using the ODS statements.

SEM and HLM

SEM and HLM have been developed for different purposes. SEM was developed to model means and covariance structures among multivariate data, whereas HLM was developed to model clustered data. One of the key differences between SEM and HLM is the assumption about the independence of observations: SEM assumes the independence of observations, whereas HLM attempts to explicitly model the dependency in data structure (Curran, 2003). However, it turned out that HLM can be incorporated into conventional SEM. The key to understand how HLM can be modeled using conventional SEM is to recognize that univariate multilevel models are actually multivariate unilevel models (Mehta & Neale, 2005). That is, the variability between clusters in HLM is captured by the common factor in SEM, and the variability within clusters in HLM is captured by the unique factor in SEM. Because the number of subjects within a cluster can vary across clusters, fitting HLM in conventional SEM framework requires maximizing the likelihood function represented as the sum of individual likelihood functions. Therefore, HLM can be specified in any SEM software package supporting full information maximum likelihood (FIML) estimation (Hox, 2013). Recent versions of SEM software packages have special commands to handle complicated set-ups for implementing HLM.

Even though HLM can be incorporated into conventional SEM software packages, it is still necessary to conduct a Monte Carlo simulation study using conventional HLM software packages such as HLMwin (Raudenbush et al., 2004) because HLM has its unique characteristics such as its capability of modeling higher level models or its flexibility of handling continuous time predictor (Hox, 2013). Unlike other SEM software packages presented in this article,
however, HLMwin does not provide built-in capabilities for a Monte Carlo simulation study. In the following section, it is demonstrated how to use the R software package to automate a Monte Carlo simulation study using HLMwin. The R functions introduced here would be useful for conducting a Monte Carlo simulation study using any software package that does not provide built-in capabilities for a Monte Carlo simulation study.

**A Simulation Study Using HLMwin**

Some standard software packages, such as HLMwin, SAS PROC MIXED, and R package lme4, are available to fit models in HLM. As SAS and R provide versatile programming capabilities, simulation procedures can be easily interconnected within SAS PROC MIXED or R package lme4. For example, data sets can be generated using R code and saved as objects within R. The generated data sets or R objects can be easily passed to the functions in the package lme4 to get parameter estimates, which can also be easily summarized within R. In such a case, connecting simulation procedures is just a matter of referring variables within the same programming environment. However, doing a simulation study using a stand-alone software such as HLM poses more practical problems in implementing a simulation if the stand-alone software does not provide capabilities for a Monte Carlo simulation. In such a case, some additional programming skills are required to automate a simulation: First, before running a software to get the desired number of replications, a large number of data and input command files need to be prepared. In doing so, file names or some commands in input command files need to be changed automatically from replication to replication. Second, a stand-alone software needs to be run repeatedly with different data and input command files from replication to replication. Third, given a large number of output files, statistics of interest need to be extracted from output files to summarize results. The aforementioned skills can be implemented using some built-in R functions, which are introduced in the remaining part of this section.

**Manipulating character strings in R.** The paste() function in R can be used to change file names and some input commands from replication to replication. Basically, the paste() function combines several arguments to form a combined string. For example, the paste() function in the example below combines the string “Level1rep,” the value stored in the variable named files, and the string “.dat” to form a combined string. If the variable named files has value of 1, the combined string assigned to filename1 becomes “Level1rep1.dat.”

```r
>filename1 <- paste("Level1rep", files, ".dat", sep="")
```

The key for changing file names and input commands from replication to replication is to use a for loop index variable as one of the arguments of the paste() function. For example, if the variable named as files in the above example is a for loop index representing replications, the string assigned to filename1 is changed from replication to replication: “Level1rep1.dat,” “Level1rep2.dat,” and so on.

**Calling system commands from R.** Given a large number of data and input command files, a simulation procedure for running a stand-alone software can be automated by running the stand-alone software in batch mode. In batch mode, the software can be run using some commands at the DOS prompt without any further manual interaction of a user. Gagne and Furlow (2009) demonstrated how SAS can be used to run Mplus, LISREL, and HLM in batch mode. In their work, DOS batch files that contain some DOS commands to run those software packages in batch mode are created and then the capability of SAS that can execute DOS system commands is used to execute those DOS batch files from SAS. In this article, the shell() function in R that can execute DOS system commands from R is used to directly run HLM without creating DOS batch files. For example, the following R command runs HLM in batch mode to estimate parameters of a model specified in the input command file, which is repl1.mdm, using the data file “repl1.mdm.”

```r
>shell("hlm2 repl1.mdm repl1.hlm")
```

**The regular expression in R.** Running software in batch mode creates a large number of output files. To summarize simulation results, statistics of interest such as parameter estimates or fit indices need to be extracted from those output files. A regular expression is a useful tool for extracting statistics of interest from output files. A regular expression is a set of literal and special characters representing a pattern that a regular expression engine tries to match in input text. For example, the regular expression “ITEM[0-9]{4}” matches ITEM0001, ITEM0002, and so on, in input text because [0-9] matches any single digit number and {n} matches n occurrences of the preceding element. Another example of the regular expression, which is used in this article, is the one to match a decimal number. The regular expression “[+–]?[0-9]+\.[0-9]+” matches a decimal number in a text. Each component of the previous regular expression needs to be explained. In a regular expression, the square brackets [ ] are used to match any single character listed in the square brackets. Therefore, [+–] matches a plus or minus sign. Since ? matches the preceding element zero or more times, a decimal number without any sign is also matched by the regular expression. On the other hand, in a regular expression, + matches the preceding element one or more times. Therefore, [0-9]+ in the example matches any positive integer. Finally, \. represents a dot in a regular expression. In all, “[+–]?[0-9]+\.[0-9]+” matches 3.14, –2.8, and so on. The regular expression is supported by many programming languages (e.g., R, SAS, C++, and Java) and command line utilities in the UNIX (e.g., grep, sed, and awk) because of its usefulness in many real world applications such as web data mining. The wide applicability of the regular expression comes from...
its flexibility that can match a specific string. By using the regular expression, users can match, modify, or extract specific strings in text files. In R, several functions support the regular expression. The grep() function returns indices of lines containing a character string that is matched by a regular expression, which is useful in extracting only the lines that are of interest from text files. For example, the following R command returns indices of lines that contain the string “with robust standard errors” in a text named x.

```r
>grep("with robust standard errors", x)
```

On the other hand, the str_extract_all() function in the R package stringr extracts all strings that match a pattern from an input vector. For example, the following R command can be used to extract all decimal numbers from an input vector x.

```r
>str_extract_all(x, "[-+]?\d+\d*\d\.*\d\+\d*)")
```

By using both grep() and str_extract_all() functions, statistics of interest can be organized into a tabular format from output files.

**HLM in batch mode.** As mentioned in the previous section, a simulation procedure for running HLM can be automated by running HLM in batch mode. However, a preliminary procedure is required to run HLM in batch mode because HLM uses its own data file format, which has .mdm (multivariate data matrix) as a file extension. An mdm data file can be created from a data file for other software packages such as SPSS or from an ASCII format text file, using an mdm template file that has .mdmt as a file extension. Given an mdm format data file, HLM can be run in batch mode with an input command file (.hlm) that contains the necessary instructions to run HLM. Therefore, to run HLM in batch mode, an mdm data file needs to be created using an mdm template file (.mdmt) first and then HLM can be run in batch mode using an mdm data file (.mdm) and input command file (.hlm). An easy way to create an mdm template file (.mdmt) and input command file (.hlm) is to use the graphical user interface of the HLM software package. All the necessary information to create those files can be typed in the graphical user interface of HLM, which provides the option to create an MDM template file (.mdmt) and input command file (.hlm) based on the information.

**Example code for a simulation using HLM.** Based on the previous discussions, an R code for automating a simple simulation using HLM is presented in Appendix B. This code is for generating data sets, writing input command files, running HLM in batch mode, and extracting statistics of interest from output files. For a demonstration purpose, a simple two-level HLM with a random intercept and slope is used: $y = \beta_0 + \beta_1 x + \epsilon$, $\beta_0 = \gamma_{00} + \gamma_{01} W + \nu_0$, and $\beta_1 = \gamma_{10} + \gamma_{11} W + \nu_1$.

In the beginning of the code, the information required for a simulation needs to be set by a user: a path for the HLM software package, a path for a folder where all the relevant files are saved, the number of replications, the number of groups, the number of subjects within each group, and parameter values for a simple two-level model. Once the information is typed, the remaining part of the code does not need to be changed.

Given the information, ASCII format data files for a simple two-level model are generated. Because separate data files for each level are required to run HLM, two separate data files for Level 1 and Level 2 are generated for each replication. The write.fwf() function is used to create data files with a fixed width format, which is required by HLM in creating an mdm format data file.

As mentioned before, HLM uses mdm data files (.mdm) for an analysis. Therefore, ASCII format data files need to be converted to mdm data files using mdm template files (.mdmt). In this example, mdm template files are created by using the write() function that can write template files line by line. The paste() function within the write() function is used to change file names from replication to replication. Once mdm template files are prepared, mdm data files are created by running HLM in batch mode using the shell() function. Similarly, input command files (.hlm) can be created using the write() and paste() functions and HLM can be run in batch mode using the shell() function with an mdm data file (.mdm) and input command file (.hlm) for each replication.

Given a large number of output files after running HLM in batch model, the estimated parameters need to be extracted from those output files to summarize simulation results. In this example, the grep() and str_extract_all() functions that support the regular expression are used to extract the statistics of interest from output files. To simplify a code, the lapply() function is used to replace the for loop statement. In the code in Appendix B, output1 is a list object in R that contains the whole contents of output files as its elements. To filter out only four lines in output files that contain estimated parameters, the grep() function is used to create a list object outPut2. The str_extract_all() function is used to extract estimated parameters, which are decimal numbers, from each line to create a tabular form of data. The results of parameter estimates across all replications will be saved in the summary folder under the user-specified main folder.

**Estimation Options in Software Packages**

The software packages presented in this article provide various estimation options to handle non-normal and categorical data. In conducting a Monte Carlo simulation study, it is very important to choose appropriate estimation options depending on research questions. In this section, various estimators developed to handle non-normal and categorical data are briefly discussed, and options to specify different estimators in the software packages are also presented.

In general, parameter estimates in SEM can be obtained by optimizing a fit function. The maximum likelihood (ML), generalized least square (GLS), and unweighted least square
(ULS) estimators share the same fit function: \( F = 1/2 \text{tr}([S-\Sigma]\Sigma^{-1}) \), where \( S \), \( \Sigma \), and \( W \) represent observed covariance, model-implied covariance, and weight matrices, respectively (Bollen, 1989; Finney & DiStefano, 2006). However, the weight matrices differ across estimators: \( W_{ML} = \Sigma(\hat{\theta}) \), \( W_{GLS} = S \), and \( W_{ULS} = I \), where \( I \) represents an identity matrix. Note that the ML and GLS estimators require the multivariate normality assumption but the ULS estimator does not. Browne (1984) showed that aforementioned fit functions are special cases of a more general fit function: \( F = [s - \sigma(\hat{\theta})]W^{-1}[s - \sigma(\hat{\theta})] \), where \( S \) and \( \Sigma(\hat{\theta}) \) are the vectors of non-redundant elements of \( S \) and \( \Sigma(\hat{\theta}) \). Because the previous fit function can be considered as a weighted sum of squared residuals, the estimator based on the previous fit function is called the weighted least square (WLS) estimator. Browne (1984) also suggested using the weight matrix that is defined as the covariances of the observed sample variances and covariances. The weight matrix suggested by Browne reflects kurtosis of observed variables, and therefore the estimator can correct the violation of the normality assumption. Because this estimator does not require the multivariate normality assumption, it is usually called the asymptotically distributed free (ADF) estimator.

Despite the theoretical merit, the ADF estimator has some practical limitations. The size of the weight matrix can increase dramatically as the number of observed variable increases. As a result, the ADF estimator is computationally demanding to calculate the inversion of the weight matrix and therefore has the limitation in the number of observed variables. Also, the ADF estimator requires a large sample size for consistent and efficient estimates (Wang & Wang, 2012). Another option to handle the non-normality without the computational burden is to adjust or rescale relevant estimates from the ML estimator. Because the ML estimator is known to provide relatively accurate parameter estimates under the non-normality (Finch, West, & MacKinnon, 1997), usually the robust chi-square statistics and standard errors of parameter estimates are adjusted or rescaled to prevent increased Type I error rates.

For categorical variables, the categorical variable methodology (CVM) was introduced to incorporate the metric of the categorical variables into the ADF estimator (B. O. Muthén, 1984). In the CVM, observed categories are related to the underlying latent response variable through threshold models. However, the CVM approach is still the ADF estimator and therefore shares the same limitations with the original ADF estimator. Robust WLS estimators (B. O. Muthén, 1993) such as weighted least squares mean-adjusted (WLSM), weighted least squares mean and variance-adjusted (WLSMV), and diagonal weighted least squares (DWLS) were developed to avoid the computational burden of the ADF estimator by inverting the diagonal elements of the weight matrix instead of the full weight matrix. Also, robust WLS estimators provide rescaled chi-square statistics and standard errors. The options that need to be specified to use aforementioned estimators in different software packages are presented in Table 1.

### Table 1. Estimation Options in the Software Packages Used in This Study.

| Estimator                                | Mplus | LISREL | PROC CALIS | lavaan |
|------------------------------------------|-------|--------|------------|--------|
| Maximum likelihood                       | ML    | ML     | ML         | ML     |
| Generalized least squares                 | GLS   | GL     | GLS        | GLS    |
| Unweighted least squares                  | ULS   | UL     | ULS        | ULS    |
| Weighted least squares                    | WLS   | WL     | WLS        | WLS    |
| Robust weighted least squares (categorical)| WLSM/| DW     | DWLS       | WLSMV  |
| Robust standard error (non-normal)        | MLR/MLM | Two-step | Not applicable | MLR/MLM |

Note. Estimation methods in Mplus, LISREL, PROC CALIS, and lavaan can be specified in the ESTIMATOR=, ME=, METHOD=, and ESTIMATOR= options, respectively.

In Mplus, generating and analyzing data sets can be done within Mplus using the MODEL POPULATION and MODEL command, respectively. The generated data sets can be saved as text files using REPSAVE and SAVE options in the MONTECARLO command. Also, external data files can be used to do a simulation by using the TYPE=MONTECARLO option in DATA command. Similarly, in LISREL, the RP command can be used to fit a model to multiple data sets in a single data file, which reports multiple results in a single output file. However, unlike Mplus, summary statistics for the multiple results are not provided by the RP command. In this article, the results created by the RP command are summarized using the R functions that support regular expression. On the other hand, a Monte Carlo simulation study using some standard software packages in SEM. Built-in Monte Carlo simulation capabilities or external software packages are used to implement Monte Carlo simulation studies for Mplus, LISREL, SAS PROC CALIS, and R package lavaan. Because of the close relationship between SEM and HLM, it is also demonstrated how to automate a Monte Carlo simulation study using HLMwin using the R software package.

**Discussion**

This article is intended to provide concrete examples of a Monte Carlo simulation study using some standard software packages in SEM. Built-in Monte Carlo simulation capabilities or external software packages are used to implement Monte Carlo simulation studies for Mplus, LISREL, SAS PROC CALIS, and R package lavaan. Because of the close relationship between SEM and HLM, it is also demonstrated how to automate a Monte Carlo simulation study using HLMwin using the R software package.
package lavaan can be greatly simplified by using the R package simsem. The bind(), model(), and sim() functions in the R package simsem can be used to specify population parameters, generate data sets, and fit a model to the data sets. The generated data sets in the R package simsem can be exported to the external data files for Mplus or LISREL. The BY statement in SAS PROC CALIS is similar to the RP command in LISREL in that it can be used to do separate analyses on observations in groups defined by the variables in the BY statement. Combined with the BY statement, the ODS output statement in SAS PROC CALIS can be used to create a SAS data set containing results across all replications.

To the author's knowledge, neither internal capabilities nor external software packages for a Monte Carlo simulation study exist for a simulation using HLMwin. In such a case, some built-in functions in R can be used to automate a simulation using HLMwin. The write() and paste() functions in R can be used to write input command files for HLMwin. Given input command files, HLMwin can be run in batch mode using the shell() function that can execute DOS system commands from R. To summarize, output files, grep(), and str_extract_all() functions supporting regular expressions can be used to extract statistics of interest from output files. The regular expression is a powerful tool for extracting specific strings from input text files and could be used in extracting and summarizing results from output files of any software packages. The R codes presented in the appendices are just simple examples of using those R functions. It is hoped that the code could provide some building blocks for researchers to write their own code to automate simulation procedures.

Appendix A

Codes for a Simulation Using Mplus

A1. Mplus code for a simulation using data sets generated by Mplus itself

TITLE:
simple two factors CFA with six indicators.
data sets are generated using Mplus itself.

MONTECARLO:
  names = y1-y6;
  nobs = 500;
  nreps = 100;
  repsave = all; ! save data files for all replications
  save = sim*.dat; ! saved file names: sim1.dat,sim2.dat,etc.
  ! simlist.dat containing the list of data file names will be
  ! also generated and will be used in a simulation using
  ! SAS PROC CALIS in Appendix E.

MODEL POPULATION: ! model specification for data generation
  f1 by y1*0.3 y2*0.4 y3*0.5; ! * = for free parameter
  f2 by y4*0.6 y5*0.7 y6*0.8;
  y1-y6@1; ! @ = for fixed parameter
  f1-f2@1;

MODEL: ! model specification for data analysis
  f1 by y1*0.3 y2*0.4 y3*0.5;
  f2 by y4*0.6 y5*0.7 y6*0.8;
  y1-y6@1;
  f1-f2@1;

A2. Mplus code for a simulation using data sets generated outside Mplus

TITLE:
simple two factors CFA with six indicators.
data sets are generated outside Mplus.
DATA:
file = simMplus.dat; ! list of external data file names
! for this example, external data files are generated by
! R package simsem in Appendix C.
type = montecarlo;

VARIABLE:
  names = y1-y6;

MODEL: ! model specification for data analysis
  f1 by y1*0.3 y2*0.4 y3*0.5;
  f2 by y4*0.6 y5*0.7 y6*0.8;
  y1-y6@1;
  f1-f2@1;

Appendix B

R Codes for a Simulation Using HLM

# R codes for automating a Monte Carlo simulation using HLM
# - written by: AUTHOR (Ph.D.)
# - last update: oct 10 2013
# - the purpose of this code is to show how R package can be used
#   to automate A Monte Carlo simulation using HLM software.
#   in this code,
#     1. text files containing simulation data will be generated
#       using simple two level equation.
#     2. .mdmt files (mdm template file) will be created to convert
#       text files to .mdm files (mdm data file for HLM).
#     3. shell() function in R is used to run HLM in batch mode
#       to convert text files to .mdm files using .mdmt files.
#     4. .hlm files (input command files) will be created to run HLM
#     5. shell() function in R is used to run HLM in batch mode
#       to estimate parameters using .HLM files.
#     6. from output files, parameter estimates of interest are
#       extracted using the functions in R that support regular
#       expression.
#   - usage
#     1. this script require two R packages. please install
#        ‘gdata’ and ‘stringr’ packages before running this code.
#     2. then CHANGE SETTINGS in “simulation setting” section below.
#     3. then select the whole code and run. all the relevant files
#        and summarized results will be saved under the ‘mainfolder’
#        that is specified by a user in the “simulation setting”.

#########################################################################
###             simulation setting               ###
#########################################################################
# simulation setting
# all generated data, input commands, and results
# will be saved in the main folder specified by a user
mainfolder <- "C:/hlm_example"
HLM2Path <- "C:\Progra~1\hlm6\"       # path for HLM software
nReplications <- 10                   # # of replications
nSubWithinGroup <- 100                 # # of subjects within each group
nGroup <- 20                           # # of groups
# model parameters setting
# in this example, data were generated based
# on the following equation:
# Level1 : Y = b0 + b1*x + r0
# Level2 : b0 = gamma00+gamma01*w+u0
#     b1 = gamma10+gamma11*w+u1
gamma00 <- 1
gamma01 <- 1
gamma10 <- 1
gamma11 <- 1
tau00 <- 1
tau11 <- 1

#########################################################################

# load library
library(gdata)   # to use write.fwf()
library(stringr)   # to use str_extract_all()

# create directories
dir.create(file.path(mainfolder))
dir.create(file.path(mainfolder,"data"))
dir.create(file.path(mainfolder,"summary"))

# check working directory
setwd(mainfolder)
getwd()

#########################################################################

###       generating data set using multilevel equation          ###
#########################################################################

for (files in 1:nReplications) {
  # generating Level 1 data
dataLevel1 <- mat.or.vec(nGroup*nSubWithinGroup,4)
colnames(dataLevel1) <- c("Group","X","W","Y")
rowIndex <- 0
for (group in 1:nGroup) {
  u0 <- rnorm(1,mean=0,sd=1)
  u1 <- rnorm(1,mean=0,sd=1)
  w <- rnorm(1,mean=0,sd=1)
  for (sub in 1:nSubWithinGroup) {
    r0 <- rnorm(1,mean=0,sd=1)
    x <- rnorm(1,mean=0,sd=1)
    y <- (gamma00+gamma01*w+u0)+(gamma10+gamma11*w+u1)*x+r0
    rowIndex <- rowIndex + 1
    dataLevel1[rowIndex,] <- c(group,x,w,y)
  }
}

# generating Level 2 data
dataLevel2 <- unique(dataLevel1[,c("Group","W")])
filename1 <- paste("Level1rep",files,".dat",sep="")
# write.fwf() is used to export data using fixed width format
formatInfoL1 <- write.fwf(data.frame(round(dataLevel1,4)),sep="",
file=file.path(mainfolder,"data",filename1)
,colnames=FALSE,formatInfo=TRUE)
filename2 <- paste("Level2rep",files,".dat",sep="")
formatInfoL2 <- write.fwf(data.frame(round(dataLevel2,4)),sep="" ,file=file.path(mainfolder,"data",filename2) ,colnames=FALSE,formatInfo=TRUE)
}

#########################################################################
###      creating .mdmt file to convert ASCII files to .mdm         ###
#########################################################################
# write() function is used to write each line of .mdmt file.
# append = TRUE is used to append a line to a file
for (rep in 1:nReplications) {
  # .mdmt file names would be rep1.mdmt, rep2.mdmt, etc.
  mdmtFile <- paste("rep",rep,".mdmt",sep="")
  write("#HLM2 MDM CREATION TEMPLATE" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("growthmodel:n" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("rawdattype:ascii" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write(paste("l1fname:\",file.path(mainfolder,"data") ,"/Level1rep",rep,".dat",sep="") ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write(paste("l2fname:\",file.path(mainfolder,"data") ,"/Level2rep",rep,".dat",sep="") ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("numl1var:3" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("numl2var:1" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("l1format:(A2,3F7.4)" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("l2format:(A2,F7.4)" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("l1missing:n" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write(paste("mdmname:rep",rep,".mdm",sep="") ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("*begin l1vars" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("X" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("W" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("Y" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("*end l1vars" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("*begin l2vars" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("W" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("*end l2vars" ,file=file.path(mainfolder,"data",mdmtFile), append=T)
run HLM in batch mode to convert .txt to .mdm setwd(file.path(mainfolder,"data")) for (rep in 1:nReplications) {
  shell(paste(HLM2Path,"hlm2"," -R rep",rep,".mdm",sep=""))
}

creating .hlm file to run HLM for (rep in 1:nReplications) {
  mdmtFile <- paste("rep",rep,".hlm",sep="")
  write(paste("#WHLM CMD FILE FOR ",file.path(mainfolder,"data","rep"),
  ,".mdm",sep=""),file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("nonlin:n",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("numit:100",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("stopval:0.0000010000",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("level1:Y=INTRCPT1+X+RANDOM",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("level2:INTRCPT1=INTRCPT2+W+random/",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("level2:X=INTRCPT2+W+random/",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("fixtau:3",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("lev1ols:10",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("accel:5",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("level1weight:none",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("level2weight:none",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("varianceknown:none",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("hypoth:n",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("resfil1:n",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("resfil2:n",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("homvar:n",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("constrain:N",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("heterol1var:n",
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write(gsub("/\",\\\",paste("graphgammas:","file.path(mainfolder,"data","graphrep"),rep,".geq",sep="")),
  ,file=file.path(mainfolder,"data",mdmtFile), append=T)
  write("lvr:n")
Appendix C

R Codes for a Simulation Using R Package simsem and lavaan

# simple two factors CFA with six indicators
library(simsem) # load simsem library

# define LY = factor loading matrix
loading <- matrix(0, 6, 2) # define 6 by 2 loading matrix with zeros
loading[1:3, 1] <- NA # NA = free parameter to be estimated
loading[4:6, 2] <- NA # 0 = fixed to zero

loadingValues <- matrix(0, 6, 2) # define population factor loadings
loadingValues[1,1] <- 0.3
loadingValues[2,1] <- 0.4
loadingValues[3,1] <- 0.5
loadingValues[4,2] <- 0.6
loadingValues[5,2] <- 0.7
loadingValues[6,2] <- 0.8

LY <- bind(loading, loadingValues) # create LY using bind()

# define PS = residual variance-covariance matrix among factors
latentCov <- matrix(NA, 2, 2) # 2 by 2 matrix, NA = free parameter
diag(latentCov) <- 1 # variances two factors are fixed to 1

PS <- binds(latentCov,0)  # create PS using bind()

# define TE = measurement error variance-covariance among indicators
TE <- binds(diag(6))   # residual variances of indicators = 1

# create templates for data generation and analysis using model()
CFA.Model <- model(LY = LY , PS = PS, TE = TE, modelType = “CFA”)

# run a Monte Carlo simulation with 100 replications
output<-sim(nRep=100,CFA.Model,n=500)
summary(output)

# export generated data sets with Mplus data format.
# program="LISREL” will export data with LISREL data format.
# this code will generate 100 data files with Mplus format.
# data file names: simMplus1.dat, simMplus2.dat, etc.
# simMplus.dat that contains data file names will also generated.
# simMplus.dat can be used for a Mplus simulation in Appendix A2.
exportData(nRep=100, model=CFA.Model, fileStem = “simMplus”,n=500, program="Mplus")

# simLISREL.dat contains the whole data sets in one file.
# simLISREL.dat can be used for a LISREL simulation in Appendix D.
exportData(nRep=100, model=CFA.Model, fileStem = “simLISREL” ,n=500, program="LISREL")

Appendix D

LISREL Code

DA NI=6 NO=500 RP=100
RA = ’simLISREL.dat’
MO NX = 6 NK = 2
Appendix E

Codes for a Simulation Using SAS PROC CALIS

E1. R code to combine separate data files generated from Mplus for SAS PROC CALIS

```r
# read file names from simlist.dat, which is generated by the Mplus code
# in Appendix A1.
files <- readLines("simlist.dat")
# read actual data files using file names from simlist.dat
dataSets <- lapply(files, function(x) cbind(read.table(x),x))
# merge separate data files into one big data file
dataSets2 <- data.frame(do.call("rbind",dataSets))
# give column names
colnames(dataSets2) <- c("y1","y2","y3","y4","y5","y6","rep")
# write csv file for SAS PROC CALIS
write.csv(dataSets2,"dataForSAS.csv",row.names=FALSE)
```

E2. SAS code for a simulation

```sas
/* import data file generated in Appendix E1 */
proc import datafile="C:\semSAS\dataForSAS.csv"
    out=dataForSAS dbms=csv replace;
    getnames=yes;
run;
/* data need to be sorted according to 'by' */
proc sort data=dataForSAS;
    by rep;
run;
proc calis data=dataForSAS method=ml;
    var = y1-y6;
    lineqs
        y1 = l1 f1 + e1,
        y2 = l2 f1 + e2,
        y3 = l3 f1 + e3,
        y4 = l3 f2 + e4,
        y5 = l5 f2 + e5,
        y6 = l6 f2 + e6;
    std
        f1-f2 = 1.0,
        e1-e6 = td1-td6;
    cov
        f1-f2 = ph3;
/* by statement is used to perform separate analysis */
/* on observations in group defined by the variables */
/* in the by statement. */
    by rep;
/* ods statement can be used to summarize results of */
/* separate analysis */
    ods output ParameterEstimatesStart=parms;
run;
```
Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research and/or authorship of this article.

References

Bandolos, D. L. (2006). The use of Monte Carlo studies in structural equation modeling research. In G. R. Hancock & R. O. Mueller (Eds.), Structural equation modeling: A second course (pp. 385-426). Greenwich, CT: Information Age.

Bollen, K. A. (1989). Structural equations with latent variables. New York, NY: Wiley.

Browne, M. W. (1984). Asymptotically distribution free methods for the analysis of covariance structures. British Journal of Mathematical and Statistical Psychology, 37, 62-83.

Cohen, A. S., Kane, M. T., & Kim, S. H. (2001). The precision of simulation study results. Applied Psychological Measurement, 25, 136-145.

Curran, P. J. (2003). Have multilevel models been structural equation models all along? Multivariate Behavioral Research, 38, 529-569.

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.

Fan, X., & Fan, X. (2005). TEACHER’S CORNER: Using SAS for Monte Carlo Simulation Research in SEM. Structural Equation Modeling, 12, 299-333.

Finch, J. F., West, S. G., & MacKinnon, D. P. (1997). Effects of sample size and nonnormality on the estimation of mediated effects in latent variable models. Structural Equation Modeling, 2, 87-105.

Finney, S. J., & DiStefano, C. (2006). Non-normal and categorical data in structural equation modeling. In G. R. Hancock & R. O. Mueller (Eds.), Structural equation modeling: A second course (pp. 269-314). Greenwich, CT: Information Age.

Gagne, P., & Furlow, C. F. (2009). Automating multiple software packages in simulation research for structural equation modeling and hierarchical linear modeling. Structural Equation Modeling, 16, 179-185.

Harwell, M. R., Stone, C. A., Hsu, T.-C., & Kirisci, L. (1996). Monte Carlo studies in item response theory. Applied Psychological Measurement, 20, 101-125.

Hox, J. J. (2013). Multilevel regression and multilevel structural equation modeling. In T. D. Little (Ed.), Oxford handbook of quantitative methods (pp. 281-294). Oxford, UK: Oxford University Press.

Jöreskog, K. G., & Sörbom, D. (2006). LISREL 8.8 for Windows [Computer software]. Skokie, IL: Scientific Software International.

Mehta, P. D., & Neale, M. C. (2005). People are variables too: Multilevel structural equations modeling. Psychological Methods, 10, 259-284.

Muthén, B. O. (1984). A general structural equation model with dichotomous, ordered categorical, and continuous latent variable indicators. Psychometrika, 49, 115-132.

Muthén, B. O. (1993). Goodness of fit with categorical and other nonnormal variables. In K. A. Bollen & J. S. Long (Eds.), Testing structural equation models (pp. 205-234). Newbury Park, CA: SAGE.

Muthén, L. K., & Muthén, B. O. (1998-2012). Mplus user’s guide (7th ed.). Los Angeles, CA: Author.

Narayanan, A. (2012). A review of eight software packages for structural equation modeling. The American Statistician, 66, 129-138.

Paxton, P., Curran, P. J., Bollen, K., Kirby, J., & Chen, F. (2001). Monte Carlo experiments: Design and implementation. Structural Equation Modeling, 8, 287-312.

Pornprasertmanit, S., Miller, P., & Schoemann, A. (2013). simsem: SIMulated Structural Equation Modeling. Retrieved from http://cran.r-project.org/web/packages/simsem/index.html

Raudenbush, S. W., Bryk, A. S., & Congdon, R. (2004). HLM 6 for Windows [Computer software]. Skokie, IL: Scientific Software International.

R Core Team. (2012). R: A language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing. Available from http://www.R-project.org/

Rosseel, Y. (2012). lavaan: An R package for structural equation modeling. Journal of Statistical Software, 48(2), 1-36. Retrieved from http://www.jstatsoft.org/v48/i02/

Wang, J., & Wang, X. (2012). Structural equation modeling: Applications using Mplus. West Sussex, UK: John Wiley.

Author Biography

Sunbok Lee received Ph.D. in Educational Psychology with Quantitative Methods major from the University of Georgia. Currently, the author is working as a postdoctoral researcher at the Center for Family Research in the University of Georgia.