The ‘simsem’ Package Manual

Beta version 0.1-4

Sunthud Pornprasertmanit
Patrick Miller
Alex Schoemann

Center for Research Methods and Data Analysis, University of Kansas

Contents

Latest Updated: May 12, 2012

Introduction

This R package has been developed for facilitating analysts to simulate and analyze data within the structural equation modeling (SEM) framework. This package aims to help analysts to create simulated data from their hypotheses or their analytic results from obtained data. The simulated data can be used for different purposes, such as power analysis, model fit evaluation, and planned missing design. The material in this version of the introduction will emphasize on

1. **Building simulated sampling distribution (SSD) for fit indices.** This will help researchers tailor their fit indices cutoff based on a priori alpha level. We will show how to find SSD for absolute model fit with various model specification.
2. **Power analysis.** This package will help analysts find power in their model in both parameter estimates and fit indices. They can find the power by accounting for possible missing data. In addition, this package will allow analysts to estimate power based on planned missing data.

Installing simsem package

Make sure that the version of your R program is at least 2.15.0 and you have the `lavaan` package (the default analysis package). The `lavaan` package can be installed by

```
install.packages("lavaan")
```

Next, install the package by typing this line in R:

```
install.packages("simsem", repos="http://rweb.quant.ku.edu/kran")
```

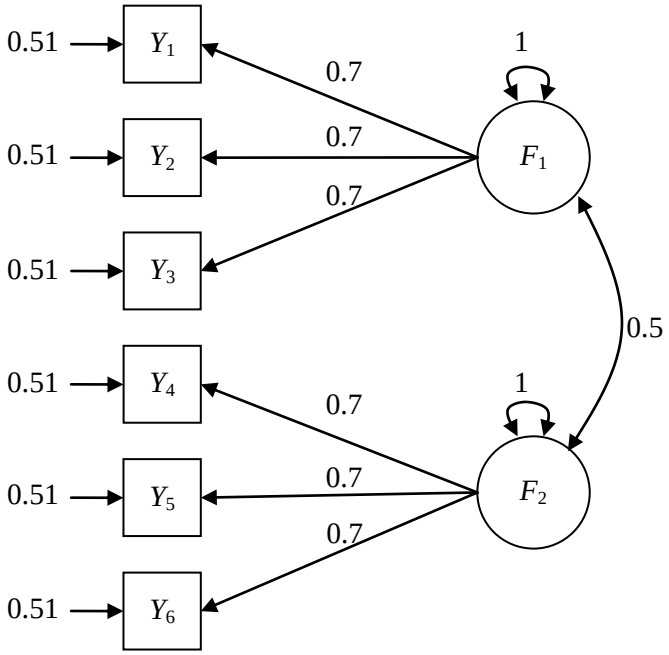
After installing the package, when you open `R` session, you can use the package by

```
library(simsem)
```

Example 1: Getting Started

Model Description

The first example is confirmatory factor analysis (CFA) model with two factors and three indicators each. Factor loadings are .7. Error variances are .51 to make the indicator variances equal to 1. Factor correlation is .5.



This package will specify a SEM model in the matrix format, like LISREL. To specify a CFA model, three matrices are required: the factor loading, the error covariance, and the factor covariance matrices. This package has general options to specify parameters. First, similar to the LISREL format, there are matrices representing covariance matrices. Second, unlike LISREL, covariance matrices are rather separated into correlation matrix and variance vector, which can be either residual variance or total variance. The second option is the useful feature in this package such that standardized parameter values can be directly specified in the code. The second option will be introduced first in this example. Then, the option for specifying the covariance matrix will be introduced in the second example.

Syntax

In creating a matrix, this program will call the matrix as matrix object. Matrix object has two components: parameters and parameter/starting values. In the parameters part, the elements of the matrix can be divided into two types: `NA` and numbers. `NA` means that the element is freely estimated in the model. Number means that the element is fixed as a specified number, usually as 0. For example, with fixed factor method for scaling identification, factor loading matrix parameters will be `NA` in elements (1,1), (2,1), (3,1), (4,2), (5,2), and (6,2). Other elements in the matrix are 0. This can be scripted in `R` as

```
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
```

The second part is the parameter values (for data generation) or starting values (for data analysis) of the free parameters. In data simulation, these parameter/starting values will be used as data generation model. The elements can be a number for a fixed parameter or a distribution object for a random parameter (which will be clarified in the third example). In this example, all parameter/starting values of factor loading matrix are 0.7. Thus, a new matrix with 6 rows and 2 columns is created and the (1,1), (2,1), (3,1), (4,2), (5,2), and (6,2) elements are specified as 0.7. The `R` script is

```
loadingValues <- matrix(0, 6, 2)
loadingValues[1:3, 1] <- 0.7
loadingValues[4:6, 2] <- 0.7
```

Next, combine two parts to create the factor loading matrix object by the simMatrix command:

```
LX <- simMatrix(loading, loadingValues)
```

If the parameter/starting values of a matrix are the same for all parameters, instead of specifying as a matrix, one parameter/starting value can be put in the simMatrix command:

```
LX <- simMatrix(loading, 0.7)
```

Users can view all specifications in a matrix object by using the summary function by

```
summary(LX)
```

In the parameter/starting values part, you will notice that if an element is not free, the parameter/starting value will be automatically set as blanks.

As explained above, the covariance matrix will be separated into two parts: a vector of error variance (or indicator variance) and error correlation. By the default of this package, indicator variances (as well as factor variances, which will be described later) are set to be 1. Thus, the factor loading can be interpreted as the standardized factor loading. The error variances by default are free parameters. From this example, the error variances are .51, which implies that indicator variances are 1 (i.e., .7×1×.7 + .51). Therefore, we will not set any error variances (or any indicator variances) and use the program default by skipping the error-variances specification and set only error correlations. There is no error correlation in this example; therefore, the error correlation is set to be identity matrix without any free parameters.

```
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
```

Because there is no free parameters in the error correlation matrix, parameter/starting values are not applicable. Next, make error correlation matrix as a symmetric matrix object by using the symMatrix function:

```
RTD <- symMatrix(error.cor)
```

The symMatrix structure is similar to the simMatrix. The main difference is that the symMatrix have more control on free parameters and constants such that the elements above and below the diagonal line are the same (i.e., symmetric). The parameter/starting values are not required in the symMatrix (as well as the simMatrix) command so there is only one attribute of the free parameters in this function.

The last matrix is the factor covariance matrix. Again, the factor covariance matrix is separated into two parts: factor variances (or factor residual variances) vector and factor correlation (or factor residual correlation). The default in this program is that the factor variances are constrained to be 1. All exogenous and endogenous factor variances are fixed parameters (i.e., fixed factor method of scale identification). Therefore, the only thing we need to specify is the factor correlation. For all correlation matrices, the diagonal elements are 1. In this model, we allow the only one element of factor correlation to be freely estimated and have the parameter/starting value of 0.5. Thus, latent correlation matrix can be specified as

```
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
```

The symmetric matrix object is created for this factor correlation by

```
RPH <- symMatrix(latent.cor, 0.5)
```

At this point, all required matrices for CFA are specified. The next step is to create an object containing the set of matrices (i.e., the factor loading matrix, the factor correlation matrix, and the error correlation matrix). This example uses CFA; therefore, the simSetCFA function will be used. The R script will be

```
CFA.Model <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD)
```

Similar to the LISREL notation, LX means the factor loading matrix, RPH means the factor correlation matrix, and RTD means the error correlation matrix. You may notice that RPH and RTD begin with 'R' to be indicated as correlation matrices. This step will apply all default set-ups of this package that is to free the error variances and to fix the factor variances. This default can be seen by the summary function:

```
summary(CFA.Model)
```

The summary function will show all parameters/starting values in the models, including all defaults. You may notice that all *X*-side in LISREL notation are changed to the *Y*-side notation automatically. The simSetCFA can be specified as the *Y*-side also as

```
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
```

This set of all CFA matrices will be used to create a data-generation object (a data object) and an analysis-model object (a model object) in order to create a set of simulated data and analyze the set of simulated data later. The data and model objects do not need to have the same set of matrices (e.g., CFA). However, in this example, I will use the same set of matrices, which is the CFA model with two factors with three indicators each without any additional constraints, in both data and model objects.

First, the data object can be specified by the simData function:

```
SimData <- simData(CFA.Model, 200)
```

The first argument is the matrix set. The second argument is a desired sample size, which is 200 in this example. You can see the specification of the data object by the summary function as well. From this, you are ready to simulate data by using the run command:

```
run(SimData)
```

You may save this data by

```
Sample <- run(SimData)
```

Next, the model object can be specified by the simModel function:

```
SimModel <- simModel(CFA.Model)
```

This program will be run by various packages. In this version of this program, the model can be only run by the lavaan package, which is the default of this program. You may see the specification of this model object by the summary function also. You may run the saved data by this model object by the run function:

```
out <- run(SimModel, Sample)
```

The result can be summarized by

```
summary(out)
```

The simulated data was analyzed by the specified CFA model. All fit indices, parameter estimates, standard errors, and Wald statistics will be provided in the screen. Finally, we need to use the data object and the model object to create simulated sampling distribution. The result object is designed to run a simulation. We can create the result object by the `simResult` function:

```
Output <- simResult(1000, SimData, SimModel)
```

The first attribute is the number of replications. The second attribute is the desired data object. The third attribute is the desired model object. After submitting this command, the program will simulate 1000 datasets and analyze all of the datasets by the specified model.

The result object contains all fit indices values that are ready for creating the SSD. You can find a fit indices cutoff based on the percentile point of the SSD. For example, we wish to find the 95th percentile (alpha level = .05). The `getCutoff` function can be used by

```
getCutoff(Output, 0.05)
```

The first argument is the result object. The second argument is the alpha level. You can see the SSD with the cutoffs in a set of figures by

```
plotCutoff(Output, 0.05)
```

The result object is set in a specific seed number. Therefore, the SSD is expected to be the same. The seed number could be changed by adding the `seed` argument in the `simResult` function as

```
Output <- simResult(1000, SimData, SimModel, seed=751785)
```

If users who have a computer with multiple processors, this package can ask R to run with multiple processors by setting the `multicore` argument as `TRUE`:

```
 Output <- simResult(1000, SimData, SimModel, multicore=TRUE)
```

The default is to use the maximum numbers of the processors in the machine. The users can specify their desired number of processors by adding the `numProc` argument as

```
Output <- simResult(1000, SimData, SimModel, multicore=TRUE, numProc=2)
```

The `summary` of the result object can be asked by

```
summary(Output)
```

The summary on the screen has mainly two sections: the fit indices cutoffs based on each alpha level and the summary of parameter estimates and standard errors. For the cutoffs, not that the larger the alpha level, the more lenient the cutoffs are. For the parameter estimates and the standard errors, there are seven columns provided:

1) `Estimate.Average`: Average of parameter estimates
2) `Esitimate.SD`: Standard deviation of parameter estimates
3) `Average.SE`: Average of standard errors of each parameter estimate
4) `Power`: The proportion of significant parameter estimates
5) `Average.Param`: Parameter values underlying simulated data
6) `Average.Bias`: Average bias of parameter estimates
7) `Coverage`: Proportion of confidence interval covered the parameter values.

Note that the columns 5-7 are not provided if users provide a list of data frame instead of data object in the simResult function (putting in the function by replacing the SimData). Also, those values in columns 5-7 have different meanings when parameters are treated as random, which are shown in the Example 4.

If users want the parameter estimates and the standard errors of all replications only, the summaryParam function can be used as

```
summaryParam(Output)
```

You might round the number in the summaryParam function by

```
round(summaryParam(Output), 3)
```

Syntax Summary

The summary of the whole script in this example is

```
1   library(simsem)
2
3   loading <- matrix(0, 6, 2)
4   loading[1:3, 1] <- NA
5   loading[4:6, 2] <- NA
6   LX <- simMatrix(loading, 0.7)
7
8   latent.cor <- matrix(NA, 2, 2)
9   diag(latent.cor) <- 1
10  RPH <- symMatrix(latent.cor, 0.5)
11
12  error.cor <- matrix(0, 6, 6)
13  diag(error.cor) <- 1
14  RTD <- symMatrix(error.cor)
15
16  CFA.Model <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD)
17  SimData <- simData(CFA.Model, 200)
18  SimModel <- simModel(CFA.Model)
19  Output <- simResult(1000, SimData, SimModel)
20  getCutoff(Output, 0.05)
21  plotCutoff(Output, 0.05)
22  summaryParam(Output)
```

Remark

1) Users may want to explicitly specify the error variances and the factor variances. This can be done by changing Lines 15-16 to

```
error.var <- rep(NA, 6)
VTD <- simVector(error.var, 0.51)

factor.var <- rep(1, 2)
VPH <- simVector(factor.var)

CFA.Model <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD, VTD = VTD, VPH = VPH)
```

   where VTD (or VTE) is the vector of the error variance and VPH (or VPS) is the vector of the factor variance

2) Users may want to include the indicators intercepts or the factor intercepts (or means) by changing Lines 15-16 to

```
intercept <- rep(NA, 6)
TX <- simVector(intercept, 0)
```

```
factor.mean <- rep(0, 2)
KA <- simVector(factor.mean)

CFA.Model <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD, TX = TX, KA = KA)
```

where TX (or TY) is the vector of the indicator intercepts and KA (or AL) is the vector of the factor intercepts

3) This program can directly specify the indicator variances (instead of the error variances) by

```
indicator.var <- rep(NA, 6)
VX <- simVector(indicator.var, 1)

CFA.Model <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD, VX = VX)
```

where VX (or VY) is the vector of the indicator variances. You cannot specify the error variances of indicators and the overall indicators variances at the same time.

4) This program can directly specify indicator means (instead of measurement intercepts) by

```
indicator.mean <- rep(NA, 6)
MX <- simVector(indicator.mean, 0)

CFA.Model <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD, MX = MX)
```

where MX (or MY) is the vector of indicator means. You cannot specify the indicator intercepts and the overall indicators means at the same time.

5) In the summaryParam function, relative bias, standardized bias, and relative bias in standard errors can be calculated by set detail as TRUE.

```
summaryParam(Output, detail=TRUE)
```

Details of how they are calculated are available in help file of the summaryParam function as

```
?summaryParam
```

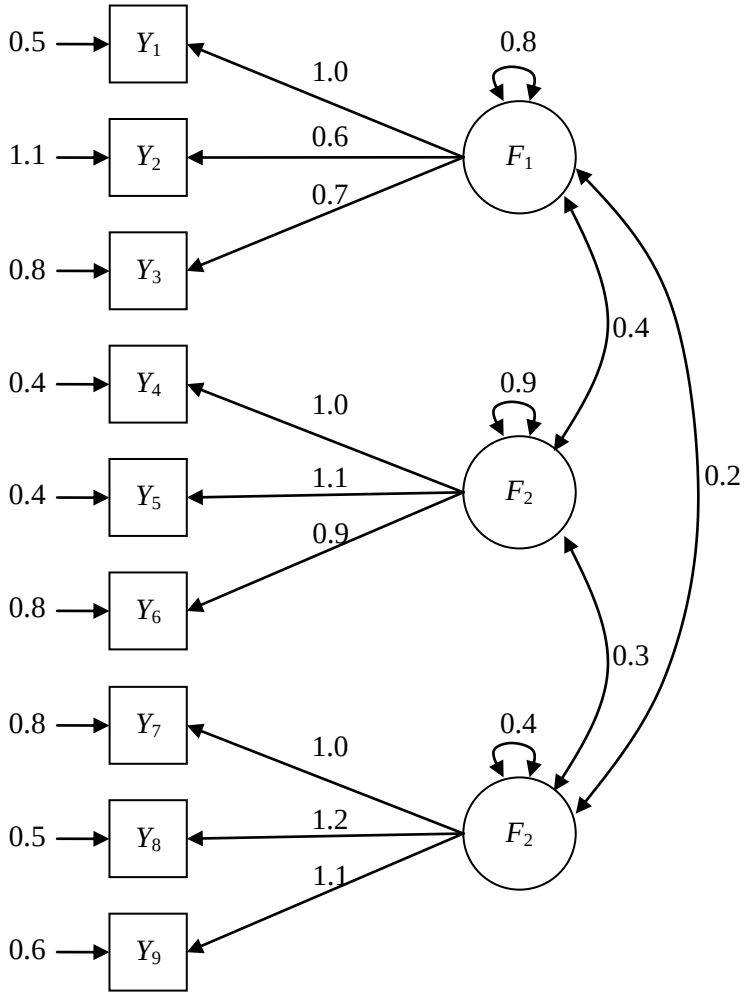
Functions Review

Functions	Usage
simMatrix	Create matrix object
symMatrix	Create symmetric matrix object
simSetCFA	Create set of matrices for CFA
simData	Create data template in order to simulate data
simModel	Create analysis model
run	Run all objects in the simsem package
summary	Summarize all objects in the simsem package
simResult	Create result of simulation
getCutoff	Get the fit indices cutoff with a priori alpha level
plotCutoff	Plot the sampling distribution of fit indices
summaryParam	Summary parameter estimates and standard errors

Example 2: Covariance Matrix Specification

Model Description

This example is also a confirmatory factor analysis (CFA) model with three factors and three indicators each. All parameter estimates listed here are similar to the values obtained from the Holzinger and Swineford (1939) example. All parameter estimates are provided in the Figure below.



The fixed factor method of scale identification is used in this example such that the first loading of each factor is fixed as 1. All factor variances are free.

Syntax

The factor loading can be specified as

```
loading <- matrix(0, 9, 3)
loading[1:3, 1] <- c(1, NA, NA)
loading[4:6, 2] <- c(1, NA, NA)
loading[7:9, 3] <- c(1, NA, NA)
loadingVal <- matrix(0, 9, 3)
loadingVal[2:3, 1] <- c(0.6, 0.7)
loadingVal[5:6, 2] <- c(1.1, 0.9)
```

```
loadingVal[8:9, 3] <- c(1.2, 1.1)
LY <- simMatrix(loading, loadingVal)
```

The parameters of the first indicator of each factor are fixed to 1. The other nonzero loadings are free. The factor covariance can be specified as

```
facCov <- matrix(NA, 3, 3)
facCovVal <- diag(c(0.8, 0.9, 0.4))
facCovVal[lower.tri(facCovVal)] <- c(0.4, 0.2, 0.3)
facCovVal[upper.tri(facCovVal)] <- c(0.4, 0.2, 0.3)
PS <- symMatrix(facCov, facCovVal)
```

All parameters in the factor covariance object are free. The `diag` function that takes a vector will create a diagonal matrix with the specified vector as the diagonal elements. The `lower.tri` and `upper.tri` functions are used to extract the elements below and above diagonal line of a matrix, respectively. Finally, the error covariance matrix can be specified as

```
errorCov <- diag(NA, 9)
errorCovVal <- diag(c(0.5, 1.1, 0.8, 0.4, 0.4, 0.8, 0.8, 0.5, 0.6))
TE <- symMatrix(errorCov, errorCovVal)
```

The diagonal elements of the error covariance are free to indicate that the error variances are estimated. Those three matrices can be combined together to represent the Holzinger and Swineford (1939) example as

```
HS.Model <- simSetCFA(LY=LY, PS=PS, TE=TE)
```

The data object, model object, and the result object can be specified by

```
SimData <- simData(HS.Model, 200)
SimModel <- simModel(HS.Model)
Output <- simResult(200, SimData, SimModel)
```

The result object can be examined by

```
getCutoff(Output, 0.05)
plotCutoff(Output, 0.05)
summary(Output)
```

Syntax Summary

The summary of the whole script in this example is

```
1   library(simsem)
2
3   loading <- matrix(0, 9, 3)
4   loading[1:3, 1] <- c(1, NA, NA)
5   loading[4:6, 2] <- c(1, NA, NA)
6   loading[7:9, 3] <- c(1, NA, NA)
7   loadingVal <- matrix(0, 9, 3)
8   loadingVal[2:3, 1] <- c(0.6, 0.7)
9   loadingVal[5:6, 2] <- c(1.1, 0.9)
10  loadingVal[8:9, 3] <- c(1.2, 1.1)
11  LY <- simMatrix(loading, loadingVal)
12
13  facCov <- matrix(NA, 3, 3)
14  facCovVal <- diag(c(0.8, 0.9, 0.4))
15  facCovVal[lower.tri(facCovVal)] <- c(0.4, 0.2, 0.3)
16  facCovVal[upper.tri(facCovVal)] <- c(0.4, 0.2, 0.3)
17  PS <- symMatrix(facCov, facCovVal)
18
19  errorCov <- diag(NA, 9)
20  errorCovVal <- diag(c(0.5, 1.1, 0.8, 0.4, 0.4, 0.8, 0.8, 0.5, 0.6))
21  TE <- symMatrix(errorCov, errorCovVal)
22
23  HS.Model <- simSetCFA(LY=LY, PS=PS, TE=TE)
```

```
24
25  SimData <- simData(HS.Model, 200)
26  SimModel <- simModel(HS.Model)
27  Output <- simResult(200, SimData, SimModel)
28  getCutoff(Output, 0.05)
29  plotCutoff(Output, 0.05)
30  summaryParam(Output)
```

Remark

1) The factor means and measurement intercepts can be also specified by changing Lines 22-23 to

```
AL <- simVector(rep(NA, 3), 0)
TY <- simVector(c(0, NA, NA, 0, NA, NA, 0, NA, NA), 0)

HS.Model <- simSetCFA(LY=LY, PS=PS, TE=TE, AL=AL, TY=TY)
```

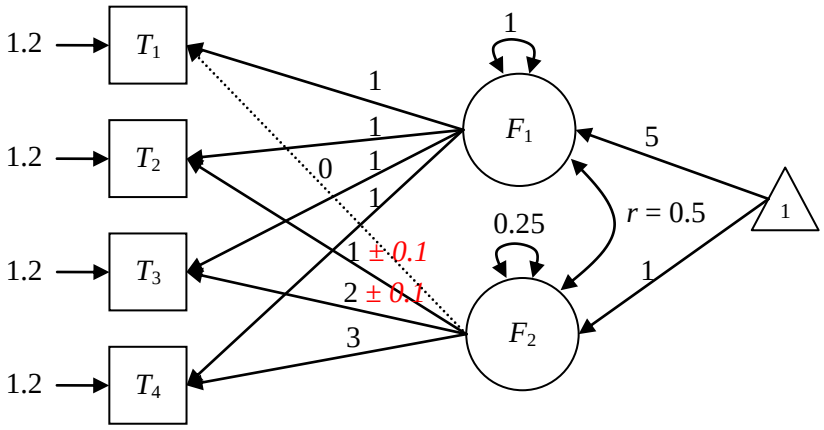
You may put the real values of factor means and measurement intercepts by putting them in the second argument of the `simVector` function.

Example 3: Model Misspecification

Model Description

In this example, we will focus on a special kind of CFA: growth curve model. The growth curve model in this example specifies two factors as intercept and slope. The factor loadings of the intercept factor are all 1. The factor loadings of the slope factor are 0, 1, 2, and 3, representing the linear change across time. In the population model, the intercept factor has the mean of 5 and the variance of 1. The slope factor has the mean of 1 and the variance of 0.25. All error variances are 1.2.

In this model, we will add a trivially misspecification in the model. In other words, we will specify a model such that the specified model is still a good approximation of the desired population. For example, the changes in the population model may not exactly follow a linear trend but the specification as a linear trend is still a good approximation of the population model. As shown in the Figure shown below, we will add a minor model misspecification that the factor loadings from the slope factor to the indicators representing Time 2 and 3 deviated from 1 and 2 by ±0.1. For example, (0, 0.9, 2.05, 3) or (0, 1.05, 1.94, 3) are the examples of the population model which is well approximated by (0, 1, 2, 3).



Syntax

The factor loading matrix can be specified as

```
factor.loading <- matrix(NA, 4, 2)
factor.loading[,1] <- 1
factor.loading[,2] <- 0:3
LY <- simMatrix(factor.loading)
```

The factor variance vector can be specified as

```
factor.var <- rep(NA, 2)
factor.var.starting <- c(1, 0.25)
VPS <- simVector(factor.var, factor.var.starting)
```

The factor correlation matrix can be specified as

```
factor.cor <- matrix(NA, 2, 2)
diag(factor.cor) <- 1
RPS <- symMatrix(factor.cor, 0.5)
```

The factor mean vector can be specified as

```
factor.mean <- rep(NA, 2)
factor.mean.starting <- c(5, 1)
AL <- simVector(factor.mean, factor.mean.starting)
```

The error variance vector can be specified as

```
VTE <- simVector(rep(NA, 4), 1.2)
```

As you can see, the `rep` function can be put directly in the argument of the function. Next, the error correlation matrix can be specified as

```
RTE <- symMatrix(diag(4))
```

The `diag` function creates an identity matrix. The attribute of the `diag` function means the number of row and columns in the identity matrix. Finally, the indicator intercepts vector can be specified as

```
TY <- simVector(rep(0, 4))
```

The CFA object that represents the growth curve model can be specified as

```
LCA.Model <- simSetCFA(LY=LY, RPS=RPS, VPS=VPS, AL=AL, VTE=VTE, RTE=RTE, TY=TY)
```

where `LY` is the factor loading matrix, `RPS` is the factor correlation matrix, `RTE` is the error correlation matrix, `VPS` is the factor variance vector, `VTE` is the error variance vector, `AL` is the factor mean vector, and `TY` is the measurement intercept vector.

As the previous example, the data, model, and result objects can be specified as

```
Data.True <- simData(LCA.Model, 300)
SimModel <- simModel(LCA.Model)
Output <- simResult(1000, Data.True, SimModel)
getCutoff(Output, 0.05)
plotCutoff(Output, 0.05)
summaryParam(Output)
```

This example uses sample size of 300 and uses 1000 replications. The next step is to add a trivially misspecification. That is, the factor loadings from the slope factor to the indicators representing Times 2 and 3 vary ±0.1. Thus, we need to make an object representing the variation, i.e., ±0.1. This object is called a distribution object. For this example, the uniform distribution with lower bound of -0.1 and upper bound of 0.1 is needed. This can be specified by the `simUnif` function as

```
u1 <- simUnif(-0.1, 0.1)
```

The first attribute is the lower bound and the second attribute is the upper bound. Other distribution is also available, such as a normal distribution (by the simNorm function). We can use this object to sample a random number from this uniform distribution by the run function.

```
run(u1)
```

You can also use summary function to see specification of this object. You may plot the distribution object by the plotDist function as

```
plotDist(u1)
```

Next, we need to put the distribution object into appropriate positions in the model. We need to put the uniform distribution object into the factor loadings from the slope factors to the indicators representing Times 2 and 3. Therefore, we need to create a factor loading matrix and put the distribution object into the factor loading matrix. Thus, the process is similar to building the simMatrix object. The only difference is to put the object name as the parameter/starting values as

```
loading.trivial <- matrix(0, 4, 2)
loading.trivial[2:3, 2] <- NA
loading.mis <- simMatrix(loading.trivial, "u1")
```

Make sure that you put single or double quotation in the parameter/starting value specification. You can use the run function to see how this matrix randomly draws numbers from the specified distribution. Because this example has the trivially misspecification in only factor loadigns, we are ready to create an object with the set of the matrices containing model misspecification, called a misspecified set object. The function name to create the misspecified set object depends on an analysis model. This example uses a simMisspecCFA function to represent the misspecification in CFA model by

```
LCA.Mis <- simMisspecCFA(LY = loading.mis)
```

You can use summary function to see the specification of this object. Let's add the trivially misspecification in the data object in misspec attribute as

```
Data.Mis <- simData(LCA.Model, 300, misspec = LCA.Mis)
```

The parameters from the misspecification set will added on top of the real parameters and then data will be created based on the combined parameters. You may use the run function on this object to create data from the population with trivially misspecification. We retain the same analysis model; therefore, we do need to change the model object. Finally, we are ready to create a new result object and examine the results of the simulation by

```
Output.Mis <- simResult(1000, Data.Mis, SimModel)
getCutoff(Output.Mis, 0.05)
plotCutoff(Output.Mis, 0.05)
summaryParam(Output.Mis)
```

You may notice that the fit indices cutoff from the simulation result without the trivially misspecification is a little more stringent than from the simulation result with the trivially misspecification.

Syntax Summary

The summary of the whole script in this example is

```
1  library(simsem)
```

```
2
3   factor.loading <- matrix(NA, 4, 2)
4   factor.loading[,1] <- 1
5   factor.loading[,2] <- 0:3
6   LY <- simMatrix(factor.loading)
7
8   factor.mean <- rep(NA, 2)
9   factor.mean.starting <- c(5, 1)
10  AL <- simVector(factor.mean, factor.mean.starting)
11
12  factor.var <- rep(NA, 2)
13  factor.var.starting <- c(1, 0.25)
14  VPS <- simVector(factor.var, factor.var.starting)
15
16  factor.cor <- matrix(NA, 2, 2)
17  diag(factor.cor) <- 1
18  RPS <- symMatrix(factor.cor, 0.5)
19
20  VTE <- simVector(rep(NA, 4), 1.2)
21
22  RTE <- symMatrix(diag(4))
23
24  TY <- simVector(rep(0, 4))
25
26  LCA.Model <- simSetCFA(LY=LY, RPS=RPS, VPS=VPS, AL=AL, VTE=VTE, RTE=RTE, TY=TY)
27
28  SimModel <- simModel(LCA.Model)
29
30  ### Get the number sign out if you wish to run the model without misspecification
31  # Data.True <- simData(LCA.Model, 300)
32  # Output <- simResult(1000, Data.True, SimModel)
33  # getCutoff(Output, 0.05)
34  # plotCutoff(Output, 0.05)
35  # summaryParam(Output)
36
37  u1 <- simUnif(-0.1, 0.1)
38
39  loading.trivial <- matrix(0, 4, 2)
40  loading.trivial[2:3, 2] <- NA
41  loading.mis <- simMatrix(loading.trivial, "u1")
42
43  LCA.Mis <- simMisspecCFA(LY = loading.mis)
44
45  Data.Mis <- simData(LCA.Model, 300, misspec = LCA.Mis)
46
47  Output.Mis <- simResult(1000, Data.Mis, SimModel)
48  getCutoff(Output.Mis, 0.05)
49  plotCutoff(Output.Mis, 0.05)
50  summaryParam(Output.Mis)
```

Remark

1) Click here to go to the list of other distribution objects.
2) We can compute the skewness and excessive kurtosis of the distribution object by the skew and kurtosis functions.

```
u1 <- simUnif(-0.1, 0.1)
skew(u1)
kurtosis(u1)
```

Note that the skew and kurtosis function on a vector of a variable or a data frame as well. See here for the description of skewness and excessive kurtosis.

Functions Review

Functions	Usage
simUnif	Create parameters distributed as uniform distribution

simNorm	Create parameters distributed as normal distribution
simMisspecCFA	Create set of matrices for misspecification in CFA
plotDist	Plot a distribution object

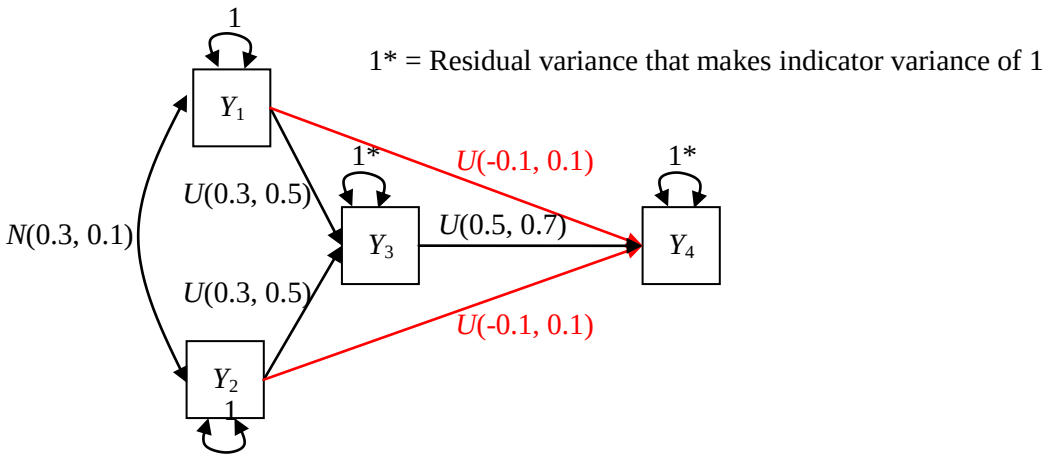
Example 4: Random Parameters

Model Description

This example changes the analysis model to path analysis model. We will show another example of adding a trivially model misspecification. In this example, the population model also has random parameters.

The hypothesized model is a full mediation model (all black paths). Users might be not sure about the exact values of the parameters in the model. Therefore, they specify parameters in ranges to represent the users' uncertainty. The effects from Y_1 to Y_3 and Y_2 to Y_3 range from 0.3 to 0.5 in a uniform distribution. The effect from Y_3 to Y_4 ranges from 0.5 to 0.7. The correlation between two exogenous variables ranges in a normal distribution with the mean of 0.3 and the standard deviation of 0.1. We need all direct effects in standardized scale. Therefore, all error variances are computed such that their indicator variances equal 1.

In the population model, the mediator does not explain all effects from the independent variables to the dependent variable. The trivially misspecification in this model is the potential direct effects from Y_1 and Y_2 to Y_4. These effects are specified in a uniform distribution with the lower and upper bounds of -0.1 and 0.1.



Syntax

First, we need to identify the distribution objects corresponding to the population and misspecification models.

```
u35 <- simUnif(0.3, 0.5)
u57 <- simUnif(0.5, 0.7)
u1 <- simUnif(-0.1, 0.1)
n31 <- simNorm(0.3, 0.1)
```

The simUnif function is used to make a uniform distribution object. The simNorm function is used to make a normal distribution object. The first and second arguments of the simUnif function are the lower and the upper bounds, respectively. The first and second arguments of the simNorm function are the mean and the standard deviation, respectively.

We will need only two matrices in this model: a path matrix and an indicator covariance matrix. The path matrix can be specified as

```
path.BE <- matrix(0, 4, 4)
path.BE[3, 1:2] <- NA
path.BE[4, 3] <- NA
starting.BE <- matrix("", 4, 4)
starting.BE[3, 1:2] <- "u35"
starting.BE[4, 3] <- "u57"
BE <- simMatrix(path.BE, starting.BE)
```

Similar to LISREL, the row number represents response variables and the column number represents predictors. For example, freeing the (3, 2) element is to estimate the regression coefficient from Y_2 to Y_3. To put random parameters, the appropriate names of the random parameter object should be set in the appropriate positions in the parameter/starting values matrix. Note that a nonrecursive (with feedback loop) model is not allowed in this program.

The indicator covariance matrix separates into the indicator variance vector and the indicator correlation (or residual correlation) matrix. First, the indicator correlation can be specified as

```
residual.error <- diag(4)
residual.error[1,2] <- residual.error[2,1] <- NA
RPS <- symMatrix(residual.error, "n31")
```

In this example, only indicator correlation between Y_1 and Y_2 is estimated. For the indicator variances, the default of this program is to make the overall indicator variances equal to 1 and all indicator variances are estimated in a path analysis model.

The matrix set of the path analysis model object can be specified by the simSetPath function as

```
Path.Model <- simSetPath(RPS = RPS, BE = BE)
```

where PS is the indicator correlation and BE is the matrix of regression coefficient.

The misspecification model in this example is in the regression coefficients only. This can be specified by the simMisspecPath function as

```
mis.path.BE <- matrix(0, 4, 4)
mis.path.BE[4, 1:2] <- NA
mis.BE <- simMatrix(mis.path.BE, "u1")
Path.Mis.Model <- simMisspecPath(BE = mis.BE)
```

Notice that the "u1" object (i.e., the uniform distribution object ranging from -0.1 to 0.1) is put in the elements (4, 1) and (4, 2) of the regression coefficient matrix to represent the misspecified direct effects.

The data object with trivially misspecification, the model object, and the result object can be created by

```
Data.Mis <- simData(Path.Model, 500, misspec = Path.Mis.Model)
SimModel <- simModel(Path.Model)
Output <- simResult(1000, Data.Mis, SimModel)
```

```
getCutoff(Output, 0.05)
plotCutoff(Output, 0.05)
summary(Output)
summaryParam(Output)
```

This example uses sample size of 500 and uses 1000 replications.

Note that the printout from the summary and summaryParam functions provides a slightly different output. There are nine columns in the parameter estimates and standard errors section. The first four columns are the same meanings as previous examples. The last five columns meanings are

5) Average.Param: The average of random parameter values underlying the simulated data across all replications
6) SD.Param: The standard deviation of the random parameter values
7) Average.Bias: The average bias of the parameter estimates from the random parameters of each replication.
8) SD.Bias: The standard deviation of the bias of all parameter estimates. This value is expected to be equal to the average of standard errors across all replications if random parameters are specified.
9) Coverage: Proportion of confidence interval covered the random parameter values underlying data in each replication.

This printout is shown only when random parameters are specified.

Syntax Summary

```
1   library(simsem)
2
3   u35 <- simUnif(0.3, 0.5)
4   u57 <- simUnif(0.5, 0.7)
5   u1 <- simUnif(-0.1, 0.1)
6   n31 <- simNorm(0.3, 0.1)
7
8   path.BE <- matrix(0, 4, 4)
9   path.BE[3, 1:2] <- NA
10  path.BE[4, 3] <- NA
11  starting.BE <- matrix("", 4, 4)
12  starting.BE[3, 1:2] <- "u35"
13  starting.BE[4, 3] <- "u57"
14  BE <- simMatrix(path.BE, starting.BE)
15
16  residual.error <- diag(4)
17  residual.error[1,2] <- residual.error[2,1] <- NA
18  RPS <- symMatrix(residual.error, "n31")
19
20  Path.Model <- simSetPath(RPS = RPS, BE = BE)
21
22  mis.path.BE <- matrix(0, 4, 4)
23  mis.path.BE[4, 1:2] <- NA
24  mis.BE <- simMatrix(mis.path.BE, "u1")
25  Path.Mis.Model <- simMisspecPath(BE = mis.BE)
26
27  Data.Mis <- simData(Path.Model, 500, misspec = Path.Mis.Model)
28  SimModel <- simModel(Path.Model)
29  Output <- simResult(1000, Data.Mis, SimModel)
30  getCutoff(Output, 0.05)
31  plotCutoff(Output, 0.05)
32  summaryParam(Output)
```

Remark

1) We can directly specify the indicator variances by changing Lines 19-20 to

```
VE <- simVector(rep(NA, 4), 1)

Path.Model <- simSetPath(RPS = RPS, BE = BE, VE = VE)
```

where VE is the vector of the indicator variances (In SEM model, VE means the overall variances of factors). You cannot specify the error variances (VPS) of indicators and the overall indicators variances at the same time.

2) This program can directly specify the indicator means (instead of the measurement intercepts) by

```
ME <- simVector(rep(NA, 4), 0)

Path.Model <- simSetPath(RPS = RPS, BE = BE, ME = ME)
```

where ME is the vector of the indicator means (In SEM model, ME means the overall means of factors). You cannot specify the indicator intercepts (AL) and the overall indicators means at the same time.

3) This program can analyze both *X* and *Y* sides at the same time. The script in Lines 8-25 can be changed to

```
path.GA <- matrix(0, 2, 2)
path.GA[1, 1:2] <- NA
GA <- simMatrix(path.GA, "u35")

path.BE <- matrix(0, 2, 2)
path.BE[2, 1] <- NA
BE <- simMatrix(path.BE, "u57")

exo.cor <- matrix(NA, 2, 2)
diag(exo.cor) <- 1
RPH <- symMatrix(exo.cor, "n31")

RPS <- symMatrix(diag(2))

Path.Model <- simSetPath(RPS = RPS, BE = BE, RPH = RPH, GA = GA, exo=TRUE)

mis.path.GA <- matrix(0, 2, 2)
mis.path.GA[2, 1:2] <- NA
mis.GA <- simMatrix(mis.path.GA, "u1")
Path.Mis.Model <- simMisspecPath(GA = mis.GA, exo=TRUE)
```

Similar to LISREL notation, we use GA for the effects from exogenous indicators to endogenous indicators, RPH for the correlations (instead of covariance) among exogenous indicators, BE for the directional effects among endogenous indicators, and RPS for the correlations among endogenous residuals.

4) Users might wish to create a dataset and would like to see the population values underlying the specific dataset. Then, the data output object can be created instead. This could be created by setting the dataOnly argument equal FALSE, as

```
dat <- run(Data.Mis, dataOnly = FALSE)
```

The data can be analyzed as usual by the run command with the model object as the first argument and the data output object as the second argument

```
out <- run(SimModel, dat)
summary(out)
```

```
summaryParam(out)
```

Notice that the output having three additional columns: the parameters underlying the data (Param), the difference between the parameters values and the parameter estimates (Bias), and whether the confidence interval covers the parameter value (Coverage). Note that this printout will be provided only when the parameter set using for data simulation and the parameter set in the analysis model are the same.

Functions Review

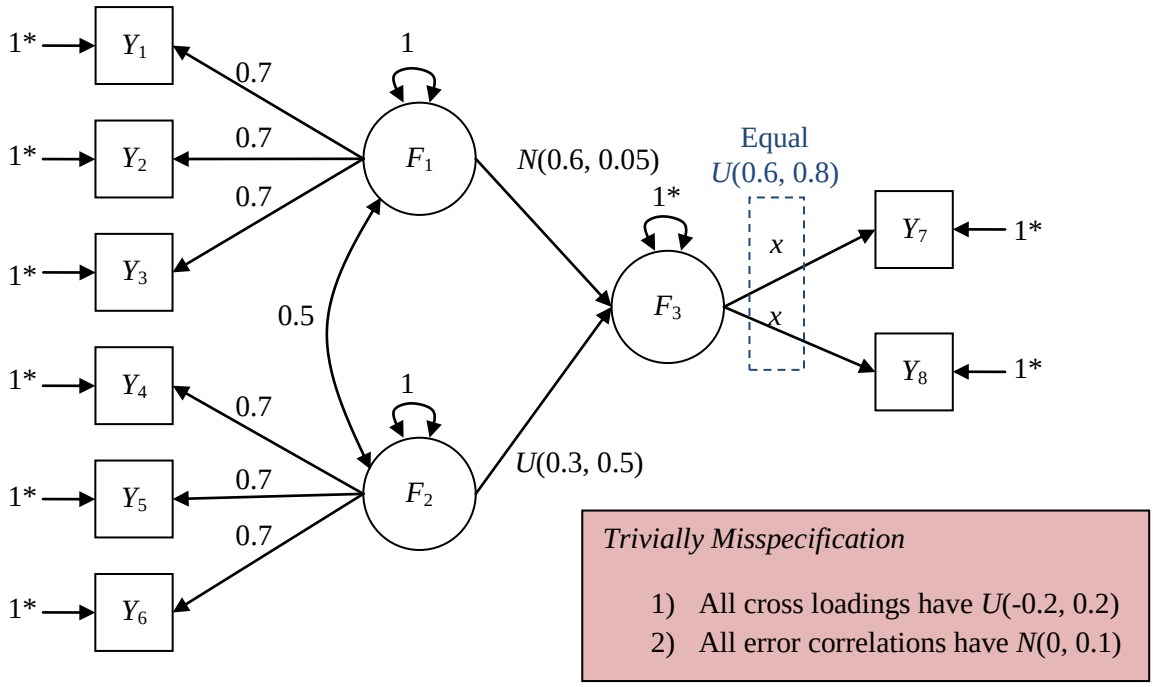
Functions	Usage
simSetPath	Create set of matrices for path analysis
simMisspecPath	Create set of matrices for misspecification in path analysis

Example 5: Equality Constraint

Model Description

This example will show how to specify full SEM model with random parameters and a trivially model misspecification. Furthermore, this example will illustrate how to set an equality constraint. The specification of the factors and indicators in the exogenous side is similar to the syntax provided in the Example 1. In this example, we will add trivial cross-loadings as a trivial misspecification. We still need to make sure that the indicator variances are still equal to 1. These two exogenous factors predict one endogenous factor. The effect from the first factor is normally distributed with the mean of 0.6 and the standard deviation of 0.05. The effect from the second factor is uniformly distributed with the lower and the upper bounds of 0.3 and 0.5. The endogenous factor has two indicators. The factor loadings are equally constrained. The parameter of the endogenous factor loadings is uniformly distributed with the lower and the upper bounds of 0.6 to 0.8. The F_3 error variance is the value that makes the F_3 overall variance equal to 1. As a result, the regression coefficients and the factor loadings can be interpreted as standardized coefficients. However, the fixing overall variances as 1 can be done only in data generation. The analysis model will fix the F_3 error variance as 1 instead (not overall variance). Therefore, the analysis result will not provide the standardized coefficients.

We will have two types of trivially misspecification in this model. First, all possible cross loadings are in a uniform distribution from -0.2 to 0.2. Second, all possible error correlations parameters are in a normal distribution with mean of 0 and *SD* of 0.1.



1* = Residual variance that makes indicator variance of 1

Syntax

First, the distribution objects in this model are created as

```
n65 <- simNorm(0.6, 0.05)
u35 <- simUnif(0.3, 0.5)
u68 <- simUnif(0.6, 0.8)
u2 <- simUnif(-0.2, 0.2)
n1 <- simNorm(0, 0.1)
```

For a full SEM model, if we consider only *Y* side, four matrices are required: the factor loading matrix, the error covariance matrix, the factor regression coefficient matrix, and the factor residual covariance matrix. The factor loading matrix can be specified as

```
loading <- matrix(0, 8, 3)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loading[7:8, 3] <- NA
loading.start <- matrix("", 8, 3)
loading.start[1:3, 1] <- 0.7
loading.start[4:6, 2] <- 0.7
loading.start[7:8, 3] <- "u68"
LY <- simMatrix(loading, loading.start)
```

If we run the `LY` object, we will see that the loadings of the endogenous indicators are not equal. We will make them equal leter. We will leave the error variances by default (overall indicator variances = 1). The error correlation matrix is specified as

```
RTE <- symMatrix(diag(8))
```

We will also leave the factor error variances set by default (overall factor variances = 1). The factor correlation matrix is specified as

```
factor.cor <- diag(3)
factor.cor[1, 2] <- factor.cor[2, 1] <- NA
```

```
RPS <- symMatrix(factor.cor, 0.5)
```

The factor regression coefficient matrix is specified as

```
path <- matrix(0, 3, 3)
path[3, 1:2] <- NA
path.start <- matrix(0, 3, 3)
path.start[3, 1] <- "n65"
path.start[3, 2] <- "u35"
BE <- simMatrix(path, path.start)
```

Now, all matrices are set up. The simSetSEM function will be used to create the set of matrices in the SEM model as

```
SEM.model <- simSetSEM(BE=BE, LY=LY, RPS=RPS, RTE=RTE)
```

LY is the factor loading matrix, TE is the error correlation matrix, BE is the regression coefficient matrix, and RPS is the factor (residual) correlation matrix. The next step is to set the matrices in the trivial model misspecification. In this example, the factor loading and the error correlation matrices are needed. The set of misspecification matrices can be created by the simMisspecSEM function as

```
loading.trivial <- matrix(NA, 8, 3)
loading.trivial[is.na(loading)] <- 0
LY.trivial <- simMatrix(loading.trivial, "u2")

error.cor.trivial <- matrix(NA, 8, 8)
diag(error.cor.trivial) <- 1
RTE.trivial <- symMatrix(error.cor.trivial, "n1")

SEM.Mis.Model <- simMisspecSEM(LY = LY.trivial RTE = RTE.trivial)
```

Now, we will create the constraint object on two factor loadings. In a single group model as in this example, a matrix is needed for each equality constraint. The number of rows in this matrix is the number of constrained parameters in each set of equality constraint. The number of columns is two representing the row and the column of the target matrices. The row name represents the name of the target matrices. In this example, the equality constraint matrix should be

$$\begin{matrix} LY \\ LY \end{matrix}\begin{bmatrix} 7 & 3 \\ 8 & 3 \end{bmatrix}$$

This means that the element (7, 3) in LY matrix equals the element (8, 3) in LY matrix. The syntax will be

```
constraint <- matrix(0, 2, 2)
constraint[1,] <- c(7, 3)
constraint[2,] <- c(8, 3)
rownames(constraint) <- rep("LY", 2)
```

Now, the constraint object can be created from this matrix by the simEqualCon function as

```
equal.loading <- simEqualCon(constraint, modelType="SEM")
```

The argument in this function is to list all equality constraints first and put the type of analysis in the modelType argument as the last argument. The possible values of the modelType attribute are "CFA", "Path", "Path.exo", "SEM", and "SEM.exo", for each type of analysis.

The next step is to create a data object.

```
Data.Original <- simData(SEM.model, 300)
Data.Mis <- simData(SEM.model, 300, misspec=SEM.Mis.Model)
Data.Con <- simData(SEM.model, 300, equalCon=equal.loading)
```

```
Data.Mis.Con <- simData(SEM.model, 300, misspec=SEM.Mis.Model, equalCon=equal.loading)
```

Here is the list of four possible combinations to make a data object. We can put the constraint object in the `equalCon` argument. In this example, the sample size is specified as 300. The model objects with and without equality constraints are

```
Model.Original <- simModel(SEM.model)
Model.Con <- simModel(SEM.model, equalCon=equal.loading)
```

Finally, the result object can be created by any possible combinations of the data and the model objects. I will show only the most complex combination (the data object with the trivial misspecification and the equality constraint combined with the model object with the equality constraint) as

```
Output <- simResult(1000, Data.Mis.Con, Model.Con)
getCutoff(Output, 0.05)
plotCutoff(Output, 0.05)
summaryParam(Output)
```

Syntax Summary

```
1   library(simsem)
2
3   n65 <- simNorm(0.6, 0.05)
4   u35 <- simUnif(0.3, 0.5)
5   u68 <- simUnif(0.6, 0.8)
6   u2 <- simUnif(-0.2, 0.2)
7   n1 <- simNorm(0, 0.1)
8
9   loading <- matrix(0, 8, 3)
10  loading[1:3, 1] <- NA
11  loading[4:6, 2] <- NA
12  loading[7:8, 3] <- NA
13  loading.start <- matrix("", 8, 3)
14  loading.start[1:3, 1] <- 0.7
15  loading.start[4:6, 2] <- 0.7
16  loading.start[7:8, 3] <- "u68"
17  LY <- simMatrix(loading, loading.start)
18
19  RTE <- symMatrix(diag(8))
20
21  factor.cor <- diag(3)
22  factor.cor[1, 2] <- factor.cor[2, 1] <- NA
23  RPS <- symMatrix(factor.cor, 0.5)
24
25  path <- matrix(0, 3, 3)
26  path[3, 1:2] <- NA
27  path.start <- matrix(0, 3, 3)
28  path.start[3, 1] <- "n65"
29  path.start[3, 2] <- "u35"
30  BE <- simMatrix(path, path.start)
31
32  SEM.model <- simSetSEM(BE=BE, LY=LY, RPS=RPS, RTE=RTE)
33
34  loading.trivial <- matrix(NA, 8, 3)
35  loading.trivial[is.na(loading)] <- 0
36  LY.trivial <- simMatrix(loading.trivial, "u2")
37
38  error.cor.trivial <- matrix(NA, 8, 8)
39  diag(error.cor.trivial) <- 1
40  RTE.trivial <- symMatrix(error.cor.trivial, "n1")
41
42  SEM.Mis.Model <- simMisspecSEM(LY = LY.trivial, RTE = RTE.trivial)
43
44  constraint <- matrix(0, 2, 2)
45  constraint[1,] <- c(7, 3)
46  constraint[2,] <- c(8, 3)
47  rownames(constraint) <- rep("LY", 2)
48  equal.loading <- simEqualCon(constraint, modelType="SEM")
```

```
49
50  Data.Original <- simData(SEM.model, 300)
51  Data.Mis <- simData(SEM.model, 300, misspec=SEM.Mis.Model)
52  Data.Con <- simData(SEM.model, 300, equalCon=equal.loading)
53  Data.Mis.Con <- simData(SEM.model, 300, misspec=SEM.Mis.Model,
        equalCon=equal.loading)
54
55  Model.Original <- simModel(SEM.model)
56  Model.Con <- simModel(SEM.model, equalCon=equal.loading)
57
58  Output <- simResult(1000, Data.Mis.Con, Model.Con)
59  getCutoff(Output, 0.05)
60  plotCutoff(Output, 0.05)
61  summaryParam(Output)
```

Remark

1) If users wish to constrain all factor loadings within a same factor to be equal, we need multiple constraints. All three constraints are

$$LY\begin{bmatrix}1 & 1\\ 2 & 1\\ 3 & 1\end{bmatrix} \qquad LY\begin{bmatrix}4 & 2\\ 5 & 2\\ 6 & 2\end{bmatrix} \qquad LY\begin{bmatrix}7 & 3\\ 8 & 3\end{bmatrix}$$

To make the syntax, Lines 44-48 can be changed as

```
constraint1 <- matrix(1, 3, 2)
constraint1[,1] <- 1:3
rownames(constraint1) <- rep("LY", 3)
constraint2 <- matrix(2, 3, 2)
constraint2[,1] <- 4:6
rownames(constraint2) <- rep("LY", 3)
constraint3 <- matrix(3, 2, 2)
constraint3[,1] <- 7:8
rownames(constraint3) <- rep("LY", 2)
equal.loading <- simEqualCon(constraint1, constraint2, constraint3, modelType="SEM")
```

The first three arguments of the `simEqualCon` function is each equality constraint.

2) Users may wish to use both *X* and *Y* sides by changing Lines 9-48 as

```
loading.X <- matrix(0, 6, 2)
loading.X[1:3, 1] <- NA
loading.X[4:6, 2] <- NA
LX <- simMatrix(loading.X, 0.7)

loading.Y <- matrix(NA, 2, 1)
LY <- simMatrix(loading.Y, "u68")

RTD <- symMatrix(diag(6))

RTE <- symMatrix(diag(2))

factor.K.cor <- matrix(NA, 2, 2)
diag(factor.K.cor) <- 1
RPH <- symMatrix(factor.K.cor, 0.5)

RPS <- symMatrix(as.matrix(1))

path.GA <- matrix(NA, 1, 2)
path.GA.start <- matrix(c("n65", "u35"), ncol=2)
GA <- simMatrix(path.GA, path.GA.start)

BE <- simMatrix(as.matrix(0))

SEM.model <- simSetSEM(GA=GA, BE=BE, LX=LX, LY=LY, RPH=RPH, RPS=RPS, RTD=RTD, RTE=RTE, exo=TRUE)

loading.X.trivial <- matrix(NA, 6, 2)
loading.X.trivial[is.na(loading.X)] <- 0
LX.trivial <- simMatrix(loading.X.trivial, "u2")
```

```
error.cor.X.trivial <- matrix(NA, 6, 6)
diag(error.cor.X.trivial) <- 1
RTD.trivial <- symMatrix(error.cor.X.trivial, "n1")

error.cor.Y.trivial <- matrix(NA, 2, 2)
diag(error.cor.Y.trivial) <- 1
RTE.trivial <- symMatrix(error.cor.Y.trivial, "n1")

RTH.trivial <- simMatrix(matrix(NA, 6, 2), "n1")

SEM.Mis.Model <- simMisspecSEM(LX = LX.trivial, RTE = RTE.trivial, RTD = RTD.trivial, RTH =
       RTH.trivial, exo=TRUE)

constraint <- matrix(0, 2, 2)
constraint[1,] <- c(1, 1)
constraint[2,] <- c(2, 1)
rownames(constraint) <- rep("LY", 2)
equal.loading <- simEqualCon(constraint, modelType="SEM.exo")
```

`LX` is the factor loading matrix of the exogenous factors. `LY` is the factor loading matrix of the endogenous factors. `RTD` is the correlation matrix of the measurement errors among exogenous indicators. `RTE` is the correlation matrix of the measurement errors among endogenous indicators. `RTH` is the correlation matrix across the measurement errors of indicators in both exogenous side (representing rows) and endogenous side (representing columns). `RPH` is the correlation matrix among the exogenous factors. `PS` is correlation matrix among residuals of endogenous factors. `GA` is the regression coefficient matrix from exogenous factors to endogenous factors. `BE` is the regression coefficient matrix among endogenous factors. If there is only one element in a matrix (1 x 1 dimension), make sure to put the `as.matrix` function on that element so that the program recognizes the element as a matrix.

Functions Review

Functions	Usage
`simSetSEM`	Create set of matrices for SEM
`simMisspecSEM`	Create set of matrices for misspecification in SEM
`simEqualCon`	Create list of equality constraints in the model

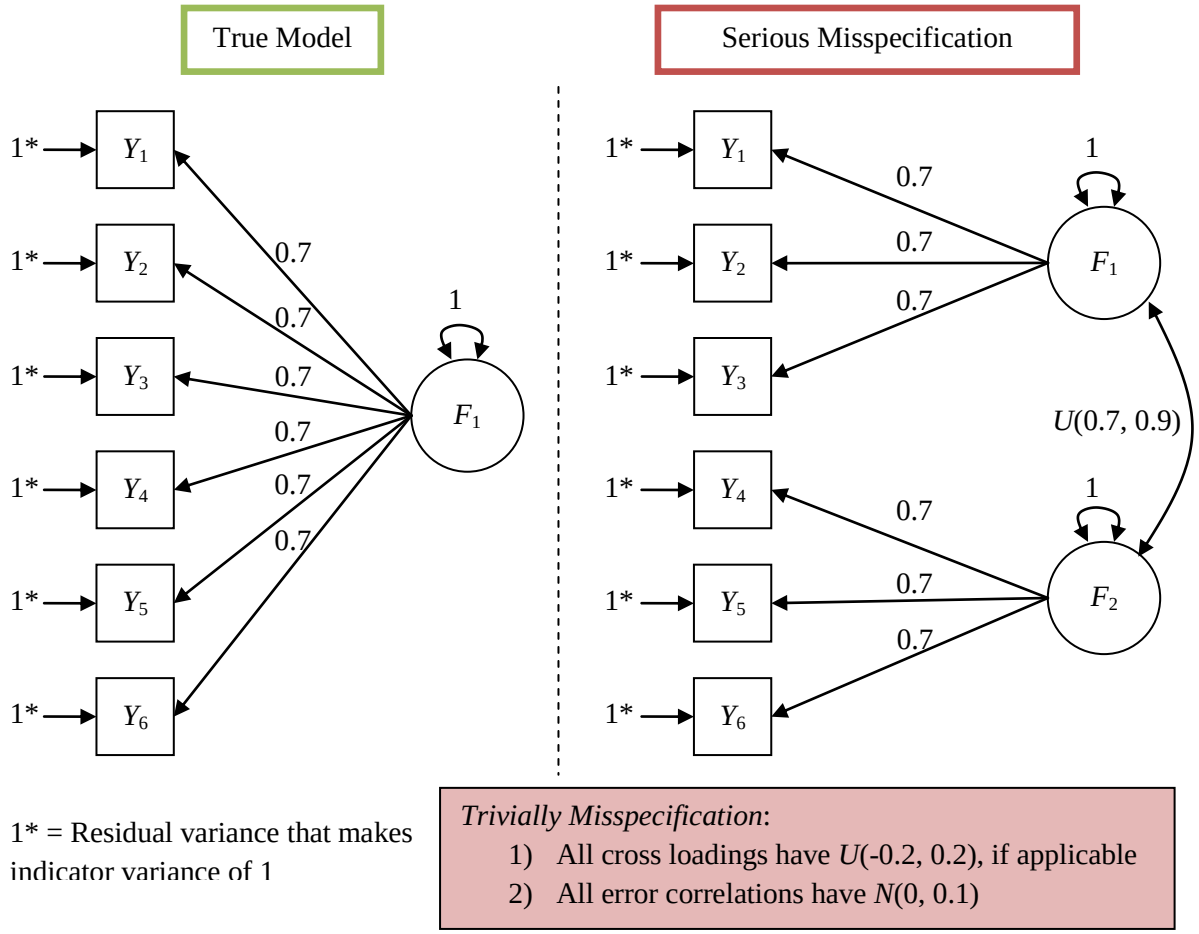
Example 6: Power Analysis in Model Evaluation

Model Description

All previous examples have shown how to find a cutoff in order to discriminate between trivial misspecification and severe misspecification using SSD. This example will show how to build two models: a correct population that users do not wish to reject and another population that users wish to reject. A cutoff is created from a correct population with a trivial model misspecification. Then, the data is created from the other population that users wish to reject. Then, we will find the proportion of data simulated from model with serious misspecification rejected by the cutoffs (i.e., statistical power).

In this example, the correct population is the one-factor model with six indicators. All factor loadings are 0.7. All error variances are calculated so that all indicator variances are 1. The trivial misspecification of the correct model includes all possible small cross-loadings and all possible small

error correlations. The other population model is a two-factor model with three indicators each. The factor correlation of the other model ranges from 0.7 to 0.8 in a uniform distribution. We assume that the two factors are not close enough to be considered as one factor that we wish to reject. Thus, we hope that the data from the two-factor model were rejected in a high proportion (high power).



1* = Residual variance that makes indicator variance of 1

Trivially Misspecification:

1) All cross loadings have $U(-0.2, 0.2)$, if applicable
2) All error correlations have $N(0, 0.1)$

Syntax

All relevant distribution objects can be specified as

```
u2 <- simUnif(-0.2, 0.2)
n1 <- simNorm(0, 0.1)
u79 <- simUnif(0.7, 0.9)
```

The correct population model can be specified as

```
loading.null <- matrix(0, 6, 1)
loading.null[1:6, 1] <- NA
LX.NULL <- simMatrix(loading.null, 0.7)
RPH.NULL <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model.NULL <- simSetCFA(LY = LX.NULL, RPS = RPH.NULL, RTE = RTD)
```

The misspecification of the correct population model can be specified as

```
error.cor.mis <- matrix(NA, 6, 6)
diag(error.cor.mis) <- 1
```

```
RTD.Mis <- symMatrix(error.cor.mis, "n1")
CFA.Model.NULL.Mis <- simMisspecCFA(RTD.Mis)
```

The result object from the correct population model with trivial misspecification can be specified as

```
SimData.NULL <- simData(CFA.Model.NULL, 500, misspec = CFA.Model.NULL.Mis)
SimModel <- simModel(CFA.Model.NULL)
Output.NULL <- simResult(1000, SimData.NULL, SimModel)
```

From here, we can find cutoffs or plot cutoffs of the correct population model. You will take a further step to create the other model as

```
loading.alt <- matrix(0, 6, 2)
loading.alt[1:3, 1] <- NA
loading.alt[4:6, 2] <- NA
LX.ALT <- simMatrix(loading.alt, 0.7)
latent.cor.alt <- matrix(NA, 2, 2)
diag(latent.cor.alt) <- 1
RPH.ALT <- symMatrix(latent.cor.alt, "u79")
CFA.Model.ALT <- simSetCFA(LY = LX.ALT, RPS = RPH.ALT, RTE = RTD)
```

We wish to reject this model. We can add a trivial misspecification in this model; however, we still wish to reject this model. We will add trivial misspecification on top of this model to broaden the range of models we wish to reject. The misspecification part can be specified as,

```
loading.alt.mis <- matrix(NA, 6, 2)
loading.alt.mis[is.na(loading.alt)] <- 0
LX.alt.mis <- simMatrix(loading.alt.mis, "u2")
CFA.Model.alt.mis <- simMisspecCFA(LY = LX.alt.mis, RTE=RTD.Mis)
```

The result object from the other model with trivial misspecification can be created by

```
SimData.ALT <- simData(CFA.Model.ALT, 500, misspec = CFA.Model.alt.mis)
Output.ALT <- simResult(1000, SimData.ALT, SimModel)
```

Note that the same model object is used and we wish that the result of the analysis will provide a bad fit index. We expect the fit indices obtained from the data from the other model indicating worse fit than the fit indices from the correct model with the trivial misspecification. Then, as previous examples, we can find the fit indices cutoffs from the correct model by

```
cutoff <- getCutoff(Output.NULL, 0.05)
```

Now, we save the cutoff in order to find power.

We can find the proportion of samples from the other model that was rejected by the cutoffs by the `getPower` function as

```
getPower(Output.ALT, cutoff)
```

The first argument is the alternative model or the model we wish to reject. The second argument is the cutoffs.

The cutoffs can be plot on a figure of overlapping histograms from the samples from both populations by the `plotPower` function as

```
plotPower(Output.ALT, Output.NULL, 0.05)
```

The first argument is the alternative model or the model we wish to reject. The second argument is the null model or the model we wish to not reject and find the cutoffs from. The third argument is the alpha level. We may set a priori cutoffs, such as RMSEA < .05, CFI > .95, TLI > .95, and SRMR < .06, and use these cutoffs to find the power by

```
cutoff2 <- c(RMSEA = 0.05, CFI = 0.95, TLI = 0.95, SRMR = 0.06)
getPower(Output.ALT, cutoff2)
plotPower(Output.ALT, cutoff2)
```

The plotPower function will plot all fit indices. If you wish to plot only some of fit indices, you can use a usedFit argument as

```
plotPower(Output.ALT, cutoff2, usedFit=c("RMSEA", "CFI"))
```

Syntax Summary

```
1   library(simsem)
2
3   u2 <- simUnif(-0.2, 0.2)
4   n1 <- simNorm(0, 0.1)
5   u79 <- simUnif(0.7, 0.9)
6
7   loading.null <- matrix(0, 6, 1)
8   loading.null[1:6, 1] <- NA
9   LX.NULL <- simMatrix(loading.null, 0.7)
10  RPH.NULL <- symMatrix(diag(1))
11  RTD <- symMatrix(diag(6))
12  CFA.Model.NULL <- simSetCFA(LY = LX.NULL, RPS = RPH.NULL, RTE = RTD)
13
14  error.cor.mis <- matrix(NA, 6, 6)
15  diag(error.cor.mis) <- 1
16  RTD.Mis <- symMatrix(error.cor.mis, "n1")
17  CFA.Model.NULL.Mis <- simMisspecCFA(RTE = RTD.Mis)
18
19  SimData.NULL <- simData(CFA.Model.NULL, 500, misspec = CFA.Model.NULL.Mis)
20  SimModel <- simModel(CFA.Model.NULL)
21  Output.NULL <- simResult(1000, SimData.NULL, SimModel)
22
23  loading.alt <- matrix(0, 6, 2)
24  loading.alt[1:3, 1] <- NA
25  loading.alt[4:6, 2] <- NA
26  LX.ALT <- simMatrix(loading.alt, 0.7)
27  latent.cor.alt <- matrix(NA, 2, 2)
28  diag(latent.cor.alt) <- 1
29  RPH.ALT <- symMatrix(latent.cor.alt, "u79")
30  CFA.Model.ALT <- simSetCFA(LY = LX.ALT, RPS = RPH.ALT, RTE = RTD)
31
32  loading.alt.mis <- matrix(NA, 6, 2)
33  loading.alt.mis[is.na(loading.alt)] <- 0
34  LX.alt.mis <- simMatrix(loading.alt.mis, "u2")
35  CFA.Model.alt.mis <- simMisspecCFA(LY = LX.alt.mis, RTE=RTD.Mis)
36
37  SimData.ALT <- simData(CFA.Model.ALT, 500, misspec = CFA.Model.alt.mis)
38  Output.ALT <- simResult(1000, SimData.ALT, SimModel)
39
40  cutoff <- getCutoff(Output.NULL, 0.05)
41  getPower(Output.ALT, cutoff)
42  plotPower(Output.ALT, Output.NULL, 0.05)
43
44  cutoff2 <- c(RMSEA = 0.05, CFI = 0.95, TLI = 0.95, SRMR = 0.06)
45  getPower(Output.ALT, cutoff2)
46  plotPower(Output.ALT, cutoff2)
```

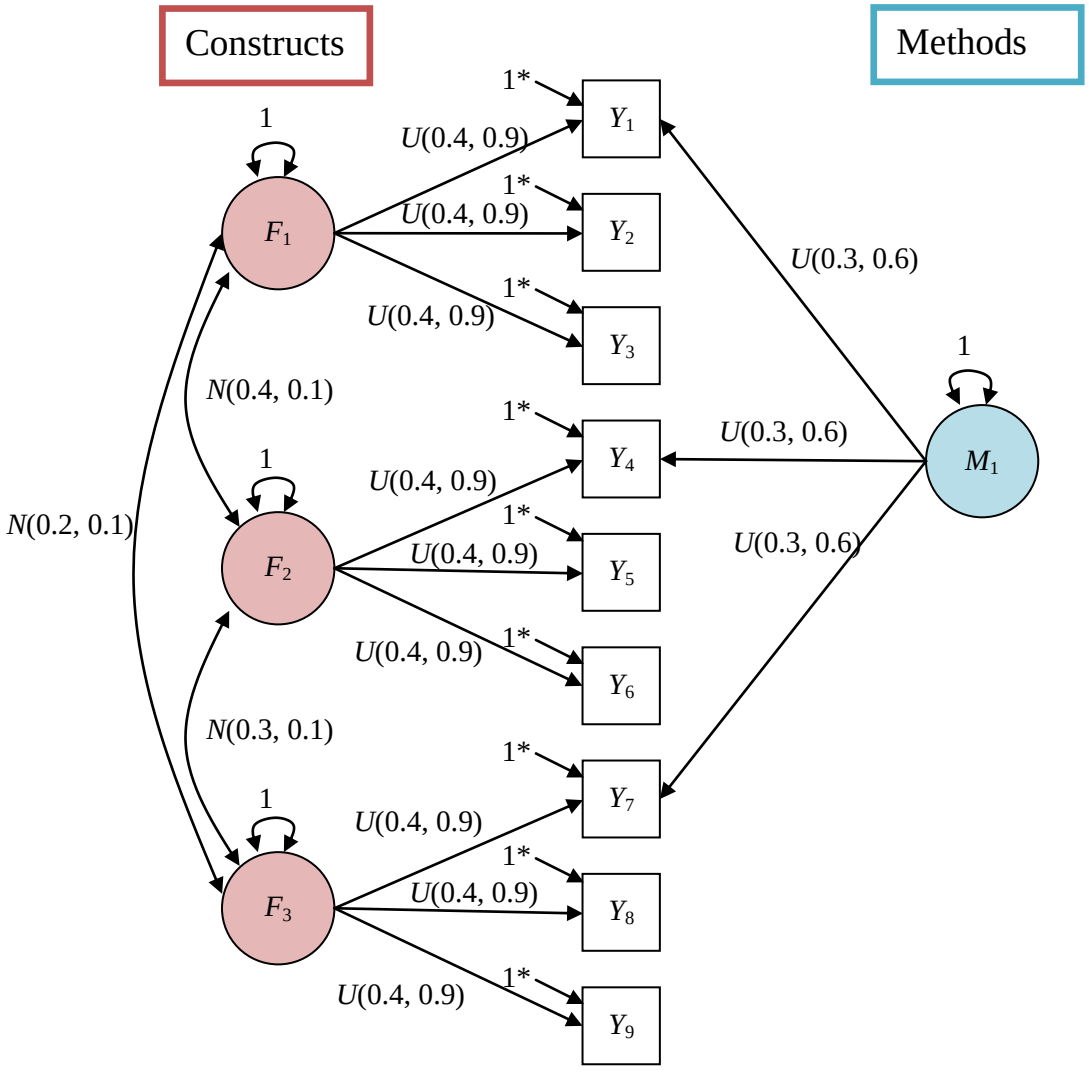
Functions Review

Functions	Usage
getPower	Get the power given cutoffs
plotPower	Visualize the power of rejection in sampling distribution

Example 7: Missing Data Handling

Model Description

This example will show how to impose missing values into datasets. The model of this example is the Multi-Trait, Multi-Method (MTMM) model. There are three traits in this model. Y_1, Y_4, and Y_7 are measured by a common method. The parameter models are shown below. The trivial model misspecification is specified in cross-loadings and error correlations. Note that the cross-loadings in the construct side are only made because the cross-loadings in the method side do not make sense. We are expected that the percentage of missing data will be approximately 20% in all variables.



1* = Residual variance that makes indicator variance of 1

Trivially Misspecification:

1) All cross loadings have U(-0.2, 0.2) only in the construct side.
2) All error correlations have N(0, 0.1)

Syntax

All relevant distribution objects can be specified as

```
u2 <- simUnif(-0.2, 0.2)
u49 <- simUnif(0.4, 0.9)
u36 <- simUnif(0.3, 0.6)
n1 <- simNorm(0, 0.1)
n21 <- simNorm(0.2, 0.1)
n31 <- simNorm(0.3, 0.1)
n41 <- simNorm(0.4, 0.1)
```

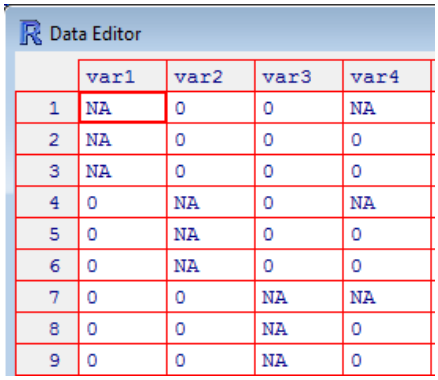
The factor loading matrix can be specified as

```
loading <- matrix(0, 9, 4)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loading[7:9, 3] <- NA
loading[c(1, 4, 7), 4] <- NA
loading.v <- matrix(0, 9, 4)
loading.v[1:3, 1] <- "u49"
loading.v[4:6, 2] <- "u49"
loading.v[7:9, 3] <- "u49"
loading.v[c(1, 4, 7), 4] <- "u36"
LY <- simMatrix(loading, loading.v)
```

For some users, type in values in a matrix might be easier. You might consider the data.entry function.

```
loading <- matrix(0, 9, 4)
data.entry(loading)
```

Then, users can edit each element of the loading matrix. The picture of the loading matrix should be



	var1	var2	var3	var4
1	NA	0	0	NA
2	NA	0	0	0
3	NA	0	0	0
4	0	NA	0	NA
5	0	NA	0	0
6	0	NA	0	0
7	0	0	NA	NA
8	0	0	NA	0
9	0	0	NA	0

The syntax of the factor correlation matrix is

```
faccor <- diag(4)
faccor[1, 2] <- faccor[2, 1] <- NA
faccor[1, 3] <- faccor[3, 1] <- NA
faccor[2, 3] <- faccor[3, 2] <- NA
faccor.v <- diag(4)
faccor.v[1, 2] <- faccor.v[2, 1] <- "n41"
faccor.v[1, 3] <- faccor.v[3, 1] <- "n21"
faccor.v[2, 3] <- faccor.v[3, 2] <- "n31"
RPS <- symMatrix(faccor, faccor.v)
```

In this example, the transpose function is used to not put the values twice. The semi-colon is used to save space. Users may use separate lines instead of the semi-colons. The factor variances are set as 1

by the program default. There is no correlation among measurement errors. The error correlation matrix can be specified as

```
RTE <- symMatrix(diag(9))
```

Thus, the MTMM model can be set up as

```
mtmm.model <- simSetCFA(LY=LY, RPS=RPS, RTE=RTE)
```

The trivial model misspecification can be specified as

```
error.cor.mis <- matrix(NA, 9, 9)
diag(error.cor.mis) <- 1
RTE.mis <- symMatrix(error.cor.mis, "n1")
loading.mis <- matrix(NA, 9, 4)
loading.mis[is.na(loading)] <- 0
loading.mis[,4] <- 0
LY.mis <- simMatrix(loading.mis, "u2")
mtmm.model.mis <- simMisspecCFA(RTE = RTE.mis, LY=LY.mis)
```

Next, we need to specify a missing object. This object will indicate both the amount of missingness imposed in the simulated data and the method to handle missing data. The missing object can be made by the `simMissing` function as

```
SimMissing <- simMissing(pmMCAR=0.2, numImps=5)
```

The `pmMCAR` argument means the proportion of values in each variable that will be imposed by missing values. The `numImps` argument is the number of imputations, which implies using the multiple imputation method in missing data handling. If the `numImps` argument is not specified, the missing data handling method will be full information maximum likelihood.

The data object and the model object can be made by

```
SimData <- simData(mtmm.model, 500, misspec = mtmm.model.mis)
SimModel <- simModel(mtmm.model)
```

We can create only one dataset, impose missing values, and analyze the data by

```
data <- run(SimData)
data <- run(SimMissing, data)
result <- run(SimModel, data, SimMissing)
summary(result)
```

The `run` function on the missing object with a dataset as the second argument will impose missing values on the data. Also, we can add the missing object on the third argument of the `run` function of the model object to specify the missing data handling method. In this example, we use multiple imputation with 5 imputations. If users do not specify the missing object in the run function, the analysis will use full information maximum likelihood by default. The summary of the result will provide two new columns: `FMI1` and `FMI2`. These are the fraction missing information using two different methods.

The result object can be specified and investigated by

```
Output <- simResult(1000, SimData, SimModel, SimMissing)
getCutoff(Output, 0.05)
plotCutoff(Output, 0.05)
summary(Output)
```

Note that the simulation could be slow because we use five copies of a dataset (i.e., multiple imputation) in each replication. Therefore, we need to run the MTMM model for five times in each replication.

The summary of the simResult object will provide four new columns: the means and the standard deviations of FMI1 and FMI2 across replications.

Syntax Summary

```
1   library(simsem)
2
3   u2 <- simUnif(-0.2, 0.2)
4   u49 <- simUnif(0.4, 0.9)
5   u36 <- simUnif(0.3, 0.6)
6   n1 <- simNorm(0, 0.1)
7   n21 <- simNorm(0.2, 0.1)
8   n31 <- simNorm(0.3, 0.1)
9   n41 <- simNorm(0.4, 0.1)
10
11  loading <- matrix(0, 9, 4)
12  loading[1:3, 1] <- NA
13  loading[4:6, 2] <- NA
14  loading[7:9, 3] <- NA
15  loading[c(1, 4, 7), 4] <- NA
16  loading.v <- matrix(0, 9, 4)
17  loading.v[1:3, 1] <- "u49"
18  loading.v[4:6, 2] <- "u49"
19  loading.v[7:9, 3] <- "u49"
20  loading.v[c(1, 4, 7), 4] <- "u36"
21  LY <- simMatrix(loading, loading.v)
22
23  faccor <- diag(4)
24  faccor[1, 2] <- faccor[2, 1] <- NA
25  faccor[1, 3] <- faccor[3, 1] <- NA
26  faccor[2, 3] <- faccor[3, 2] <- NA
27  faccor.v <- diag(4)
28  faccor.v[1, 2] <- faccor.v[2, 1] <- "n41"
29  faccor.v[1, 3] <- faccor.v[3, 1] <- "n21"
30  faccor.v[2, 3] <- faccor.v[3, 2] <- "n31"
31  RPS <- symMatrix(faccor, faccor.v)
32
33  RTE <- symMatrix(diag(9))
34
35  mtmm.model <- simSetCFA(LY=LY, RPS=RPS, RTE=RTE)
36
37  error.cor.mis <- matrix(NA, 9, 9)
38  diag(error.cor.mis) <- 1
39  RTE.mis <- symMatrix(error.cor.mis, "n1")
40  loading.mis <- matrix(NA, 9, 4)
41  loading.mis[is.na(loading)] <- 0
42  loading.mis[,4] <- 0
43  LY.mis <- simMatrix(loading.mis, "u2")
44  mtmm.model.mis <- simMisspecCFA(RTE = RTE.mis, LY=LY.mis)
45
46  SimMissing <- simMissing(pmMCAR=0.2, numImps=5)
47
48  SimData <- simData(mtmm.model, 500, misspec = mtmm.model.mis)
49  SimModel <- simModel(mtmm.model)
50
51  Output <- simResult(1000, SimData, SimModel, SimMissing)
52  getCutoff(Output, 0.05)
53  plotCutoff(Output, 0.05)
54  summary(Output)
```

Remark

1) If users wish to not impose missing values on a set of variables, they can specify the ignoreCols argument in the missing object in the Line 46 as

```
SimMissing <- simMissing(pmMCAR=0.2, numImps=5, ignoreCols=c(1, 4, 7))
```

Variables 1, 4, and 7 will not have any missing values.

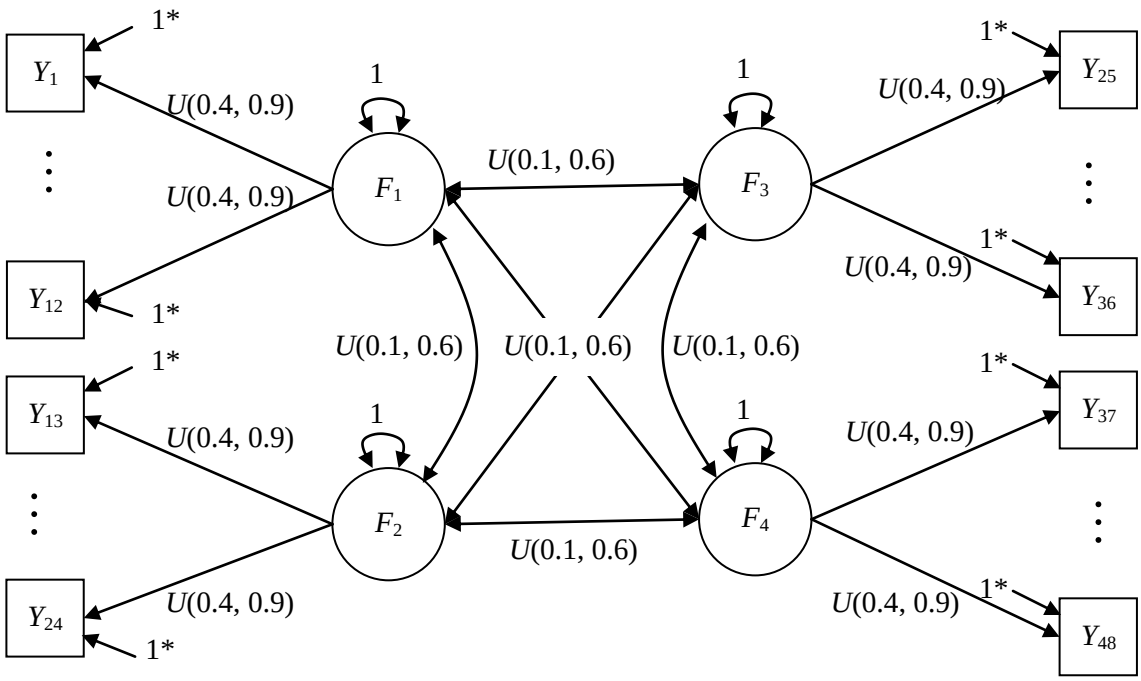
Functions Review

Functions	Usage
simMissing	Create a missing object

Example 8: Planned Missing Design

Model Description

This example will show how to implement a planned missing data. The model of this example is confirmatory factor analysis model with 4 factors and 12 indicators in each factor. We will make a three-form design such that the Indicators 1-3 in each factor are observed in all form, Indicators 4-6 are missing in Form 1, Indicators 7-9 are missing in Form 2, and Indicators 10-12 are missing in Form 3. The factor loading of all indicators are uniformly distributed from 0.4 to 0.9. The factor correlations are uniformly distributed from 0.1 to 0.6. The error variances are constrained such that the indicators variances will be equal to 1. The trivial model misspecification is specified in cross-loadings only, which is uniformly distributed from -0.2 to 0.2.



1* = Residual variance that makes indicator variance of 1

Trivially Misspecification:
All cross loadings have $U(-0.2, 0.2)$.

Syntax

All relevant distribution objects can be specified as

```
u2 <- simUnif(-0.2, 0.2)
u49 <- simUnif(0.4, 0.9)
u16 <- simUnif(0.1, 0.6)
```

The parameter model can be specified as

```
loading <- matrix(0, 48, 4)
loading[1:12, 1] <- NA
loading[13:24, 2] <- NA
loading[25:36, 3] <- NA
loading[37:48, 4] <- NA
LY <- simMatrix(loading, "u49")
faccor <- matrix(NA, 4, 4)
diag(faccor) <- 1
RPS <- symMatrix(faccor, "u16")
RTE <- symMatrix(diag(48))
CFA.model <- simSetCFA(LY=LY, RPS=RPS, RTE=RTE)
```

The trivial model misspecification can be specified as

```
loading.mis <- matrix(NA, 48, 4)
loading.mis[is.na(loading)] <- 0
LY.mis <- simMatrix(loading.mis, "u2")
CFA.model.mis <- simMisspecCFA(LY=LY.mis)
```

Next, we need to specify a missing object specifying the three-form design. We need to make a group of variables in Set X (variables without any missing values), Set 1, Set 2, and Set 3. Subjects with Form 1 will answer the variables in Set X and Set 1. Subjects with Form 2 will answer the variables in Set X and Set 2. Subjects with Form 3 will answer the variables in Set X and Set 3. After we specify the sets of variables, we group them together in a list and make a missing object as

```
setx <- c(1:3, 13:15, 25:27, 37:39)
set1 <- setx + 3
set2 <- set1 + 3
set3 <- set2 + 3
itemGroups <- list(setx, set1, set2, set3)

SimMissing <- simMissing(nforms=3, itemGroups=itemGroups, numImps=5)
```

The `nforms` argument means the number of forms in the planned missing data design. The `itemGroups` argument means the sets of variables. The number of set must be greater than the number of forms by 1. Then, the `numImps` argument is the number of imputations.

The data, model, and result objects can be made and investigated by

```
SimData <- simData(CFA.model, 1000, misspec = CFA.model.mis)
SimModel <- simModel(CFA.model)
Output <- simResult(1000, SimData, SimModel, SimMissing)
getCutoff(Output, 0.05)
plotCutoff(Output, 0.05)
summary(Output)
```

Again, the simulation could be slow because we use five copies of a dataset (i.e., multiple imputation) in each replication.

Syntax Summary

```
1  library(simsem)
2
3  u2 <- simUnif(-0.2, 0.2)
4  u49 <- simUnif(0.4, 0.9)
5  u16 <- simUnif(0.1, 0.6)
6
7  loading <- matrix(0, 48, 4)
8  loading[1:12, 1] <- NA
```

```
9   loading[13:24, 2] <- NA
10  loading[25:36, 3] <- NA
11  loading[37:48, 4] <- NA
12  LY <- simMatrix(loading, "u49")
13
14  faccor <- matrix(NA, 4, 4)
15  diag(faccor) <- 1
16  RPS <- symMatrix(faccor, "u16")
17
18  RTE <- symMatrix(diag(48))
19
20  CFA.model <- simSetCFA(LY=LY, RPS=RPS, RTE=RTE)
21
22  loading.mis <- matrix(NA, 48, 4)
23  loading.mis[is.na(loading)] <- 0
24  LY.mis <- simMatrix(loading.mis, "u2")
25  CFA.model.mis <- simMisspecCFA(LY=LY.mis)
26
27  setx <- c(1:3, 13:15, 25:27, 37:39)
28  set1 <- setx + 3
29  set2 <- set1 + 3
30  set3 <- set2 + 3
31  itemGroups <- list(setx, set1, set2, set3)
32
33  SimMissing <- simMissing(nforms=3, itemGroups=itemGroups, numImps=5)
34
35  SimData <- simData(CFA.model, 1000, misspec = CFA.model.mis)
36  SimModel <- simModel(CFA.model)
37  Output <- simResult(100, SimData, SimModel, SimMissing)
38  getCutoff(Output, 0.05)
39  plotCutoff(Output, 0.05)
40  summary(Output)
```

Remark

1) Users may implement a two-method design. For example, Indicators 1 is an expensive measurement and users wish to measure it for only 50% of all subjects. The missing object in Line 33 can be changed to

```
SimMissing <- simMissing(twoMethod=c(1, 0.5), numImps=5)
```

where the `twoMethod` argument is the specification of the two-method design. It takes a vector with two arguments: the index of variable that researchers wish to impose missing values and the proportion of missing values.

Example 9: Nonnormal Distribution

Model Description

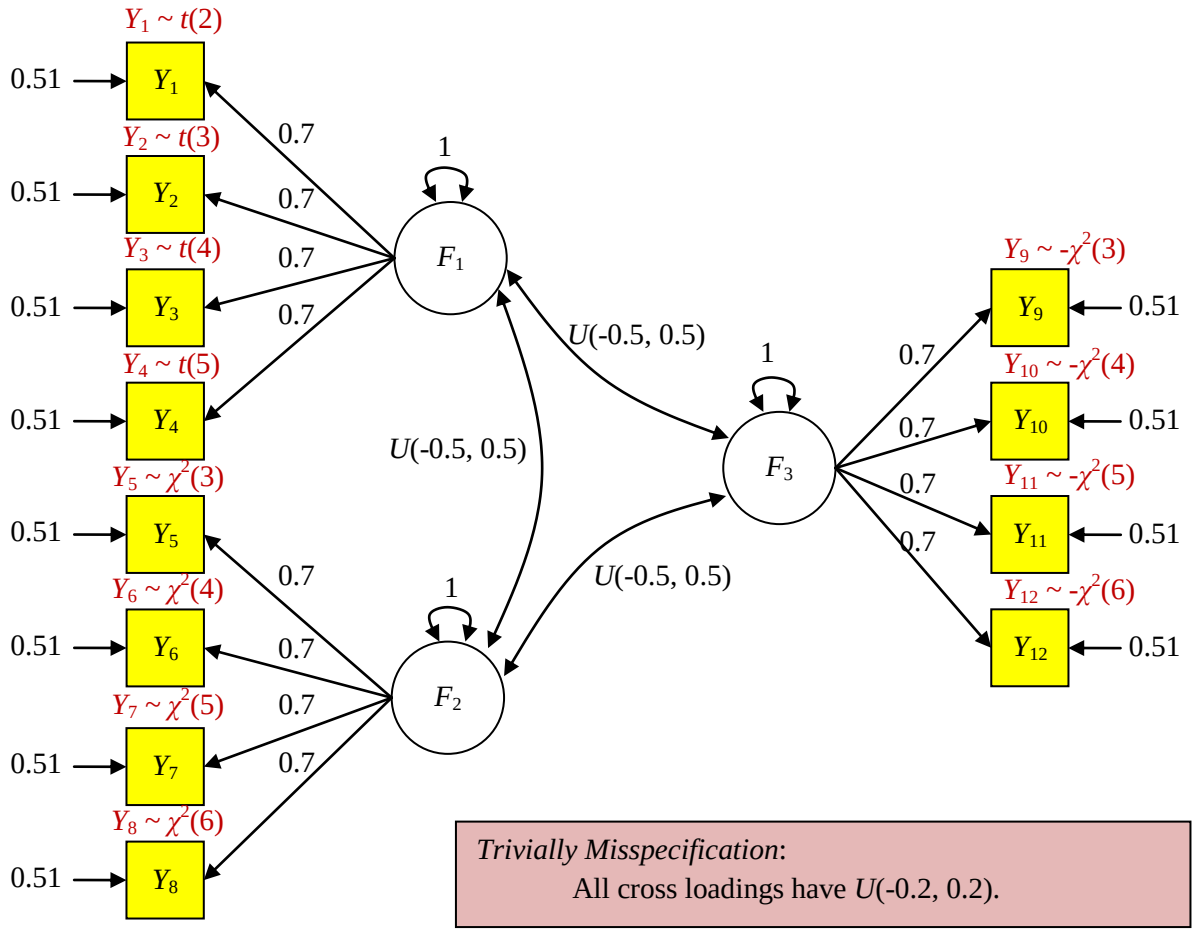
In this example and the next example, we will discuss about how to create nonnormal distributed data. We have two ways to create data and model the nonnormal distribution. The first method is to create data from the model implied means and covariance matrix among indicators. This method has been used since the first example. In this example, the nonnormal distribution is created directly from the model implied means and covariance matrix among indicators.

This package uses Gaussian copula. The underlying distribution among indicators is a multivariate normal distribution. Then, users specify the marginal distribution of each indicator. The marginal distribution can be arbitrary. Then, a phantom datapoint is generated from the underlying distribution. The cumulative probabilities of the phantom datapoint with respect to each marginal

distribution of the marginal distribution of the multivariate normal distribution are calculated. Then, the real data is the datapoint in a specified distribution that provides the same marginal cumulative probabilities of the phantom datapoint.

We will consider a CFA model with three factors and four indicators each. Factor loadings are .7. Error variances are .51. Factor correlation ranges from -0.5 to 0.5 in a uniform distribution. The model misspecification is that all cross loadings range from -0.3 to 0.3 in uniform distribution.

We will make data have high kurtosis. Let Y_1 to Y_4 have *t*-distribution with degrees of freedom from 2 to 5. Let Y_5 to Y_8 have chi-square distribution with degrees of freedom from 3 to 6. Let Y_9 to Y_{12} have chi-square distribution with degrees of freedom from 3 to 6 but flip them between right and left (i.e., from positively skewed to negatively skewed).



Syntax

All relevant distribution objects can be specified as

```
u2 <- simUnif(-0.2, 0.2)
u5 <- simUnif(-0.5, 0.5)
t2 <- simT(2)
t3 <- simT(3)
```

```
t4 <- simT(4)
t5 <- simT(5)
chi3 <- simChisq(3)
chi4 <- simChisq(4)
chi5 <- simChisq(5)
chi6 <- simChisq(6)
```

Click here to see all possible distribution objects. The simT function is the random *t* distribution object. Its argument is its degree of freedom. Its non-centrality parameter can be specified as the second argument, which is 0 by default. The simChisq function is the random chi-squared distribution object. Its argument is its degree of freedom. Its non-centrality parameter can be specified as the second argument, which is 0 by default.

The parameter model can be specified as

```
loading <- matrix(0, 12, 3)
loading[1:4, 1] <- NA
loading[5:8, 2] <- NA
loading[9:12, 3] <- NA
LX <- simMatrix(loading, 0.7)

latent.cor <- matrix(NA, 3, 3)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, "u5")

error.cor <- matrix(0, 12, 12)
diag(error.cor) <- 1
RTD <- symMatrix(error.cor)

CFA.Model <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD)
```

The trivial model misspecification can be specified as

```
loading.mis <- matrix(NA, 12, 3)
loading.mis[is.na(loading)] <- 0
LY.mis <- simMatrix(loading.mis, "u2")
CFA.model.mis <- simMisspecCFA(LY=LY.mis)
```

Next, we need to specify the distribution of indicators. We will use a data distribution object to model the indicator distribution. The data distribution object can be created by the simDataDist function and list the distribution objects as the arguments of the function as

```
SimDataDist <- simDataDist(t2, t3, t4, t5, chi3, chi4, chi5, chi6, chi3, chi4, chi5, chi6,
    reverse=c(rep(FALSE, 8), rep(TRUE, 4)))
```

The arguments, t2 to chi6, are the distribution of each variable from Y_1 to Y_{12}. The reverse argument is to flip the distribution from right and left. Because we need to flip the distributions of Y_9 to Y_{12}, we need the last four elements of the vector for the reverse attribute to be TRUE. If we wish to make the same distribution for all variables, we can put only one distribution object, the reverse argument, and specify the p argument, which is the number of variables. For example,

```
SimDataDist <- simDataDist(chi3, p=12, reverse=TRUE)
```

This is the data distribution object that all twelve variables have chi-squared distributed with degree of freedom of 3 and all variables' distributions are flipped. The summary function can be applied to the data distribution object to find a description of the object.

The data object can be accounted for the data distribution object by

```
SimData <- simData(CFA.Model, 200, misspec=CFA.model.mis, indDist=SimDataDist)
```

The addition argument is indDist that is the distribution of indicators specification.

The existence of the nonnormal distribution will violate the assumption of the maximum likelihood estimator in structural equation modeling. Therefore, other estimators might be needed. The estimator option can be specified by the estimator argument when building a model object as

```
SimModel <- simModel(CFA.Model, estimator="mlm")
```

The mlm is the maximum likelihood estimator with Satorra and Bentler scale correction. The other option can be found in the help page of the sem function from the lavaan package as

```
?sem
```

The result object can be specified and investigated by

```
Output <- simResult(1000, SimData, SimModel)
getCutoff(Output, 0.05)
plotCutoff(Output, 0.05)
summary(Output)
```

Syntax Summary

The summary of the whole script in this example is

```
1   library(simsem)
2
3   u2 <- simUnif(-0.2, 0.2)
4   u5 <- simUnif(-0.5, 0.5)
5   t2 <- simT(2)
6   t3 <- simT(3)
7   t4 <- simT(4)
8   t5 <- simT(5)
9   chi3 <- simChisq(3)
10  chi4 <- simChisq(4)
11  chi5 <- simChisq(5)
12  chi6 <- simChisq(6)
13
14  loading <- matrix(0, 12, 3)
15  loading[1:4, 1] <- NA
16  loading[5:8, 2] <- NA
17  loading[9:12, 3] <- NA
18  LX <- simMatrix(loading, 0.7)
19
20  latent.cor <- matrix(NA, 3, 3)
21  diag(latent.cor) <- 1
22  RPH <- symMatrix(latent.cor, "u5")
23
24  error.cor <- matrix(0, 12, 12)
25  diag(error.cor) <- 1
26  RTD <- symMatrix(error.cor)
27
28  CFA.Model <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD)
29
30  loading.mis <- matrix(NA, 12, 3)
31  loading.mis[is.na(loading)] <- 0
32  LY.mis <- simMatrix(loading.mis, "u2")
33  CFA.model.mis <- simMisspecCFA(LY=LY.mis)
34
35  SimDataDist <- simDataDist(t2, t3, t4, t5, chi3, chi4, chi5, chi6, chi3, chi4, chi5, chi6,
    reverse=c(rep(FALSE, 8), rep(TRUE, 4)))
36  SimData <- simData(CFA.Model, 200, misspec=CFA.model.mis, indDist=SimDataDist)
37  SimModel <- simModel(CFA.Model, estimator="mlm")
38  Output <- simResult(1000, SimData, SimModel)
39  getCutoff(Output, 0.05)
40  plotCutoff(Output, 0.05)
41  summary(Output)
```

Remark

1) The data distribution object can be plotted by the `plotDist` function. However, the data distribution object can be plotted for either one or two variables only. For two variables, the correlation between two variables is 0 by default. We can change the correlation between variables by changing the `r` argument. For example,

```
g21 <- simGamma(2, 1)
n01 <- simNorm(0, 1)

object <- simDataDist(g21)
plotDist(object)

object2 <- simDataDist(g21, n01)
plotDist(object2, r=0.5)
```

Note that `simGamma` is the constructor of the random gamma distribution object. If we have the data object distribution with multiple variables already, we can select the variables by setting the `var` argument as

```
g21 <- simGamma(2, 1)
n01 <- simNorm(0, 1)
chi2 <- simChisq(2)
obj <- simDataDist(g21, n01, chi2)
plotDist(obj, var=c(2,3))
```

2) If users wish to use the means and variances from the specified distribution instead of the means and variances implied from a specified model, we can specify the `keepScale` argument as `FALSE`. For example, change Line 35 as

```
SimDataDist <- simDataDist(t2, t3, t4, t5, chi3, chi4, chi5, chi6, chi3, chi4, chi5, chi6,
    reverse=c(rep(FALSE, 8), rep(TRUE, 4)), keepScale=FALSE)
```

Functions Review

Functions	Usage
`simDataDist`	Create data distribution object

Example 10: Nonnormal Factor Distribution

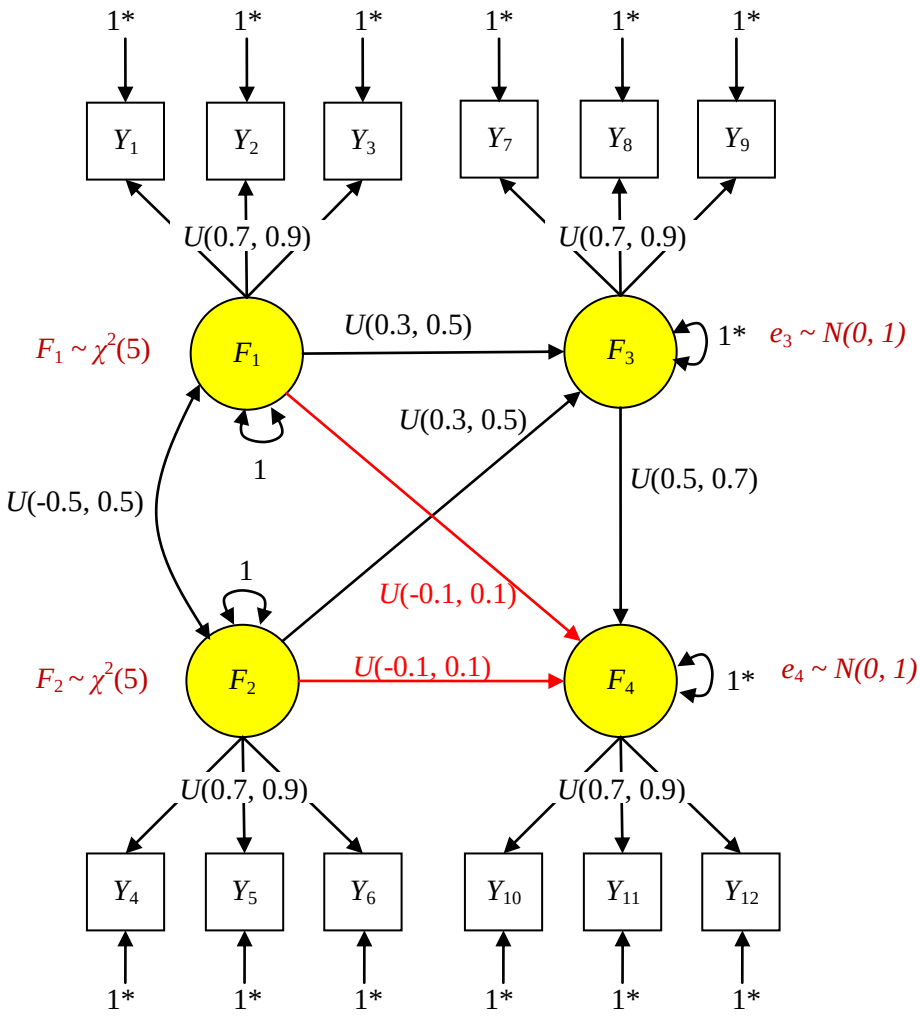
Model Description

This example will also show how to create nonnormal distribution. Instead of generating data directly from model-implied means and covariance matrix of observed indicators, this example uses a different approach, which will be referred as sequential method. This sequential method will generate all exogenous factors first, then endogenous factors, and then measurement errors. After that, the factor scores and error scores are combined together to get the observed scores. This sequential method will allow us to pinpoint the part inside the model that are not normally distributed. This example will show nonnormal exogenous factors. The nonnormality model can be built by the Gaussian copula approach.

We will consider a SEM model with four factors and three indicators each. The first two factors are exogenous and influence the fourth factor. The third factor fully mediates the direct effect from the first two factors to the fourth factor. The correlation between the first and the second factors ranges from -0.5 to 0.5 in uniform distribution. The effect from the first two factors to the third factors ranges from 0.3

to 0.5 in uniform distribution. The effect from the third factor to the fourth factor ranges from 0.5 to 0.7 in uniform distribution. All exogenous factor variances are 1 and all error variances of endogenous factors are equal to the value that makes overall factor variances equal to 1. The factor loadings range from 0.7 to 0.9 in uniform distribution. The measurement variance will be made so that overall indicators variances equal to 1.

There are three types of misspecification imposed here. First, the cross loadings ranges from -0.3 to 0.3 in uniform distribution. Second, the error correlations range in normal distribution with the mean of 0 and standard deviation of 0.1. Third, the direct effects are ranged from -0.1 to 0.1 in uniform distribution. The marginal distributions of the exogenous factors are chi-squared distribution with the degree of freedom of 5. The distributions of residuals of endogenous factors are normally distributed.



1* = Residual variance that makes indicator variance of 1

Trivially Misspecification:

1. All cross loadings have $U(-0.3, 0.3)$.
2. All error correlations have $N(0, 0.1)$.
3. All direct effects have $U(-0.1, 0.1)$

Syntax

All relevant distribution objects can be specified as

```
u35 <- simUnif(0.3, 0.5)
u57 <- simUnif(0.5, 0.7)
u1 <- simUnif(-0.1, 0.1)
u3 <- simUnif(-0.3, 0.3)
n1 <- simNorm(0, 0.1)
n31 <- simNorm(0.3, 0.1)
u79 <- simUnif(0.7, 0.9)
chi5 <- simChisq(5)
```

The parameter model can be specified as

```
path.BE <- matrix(0, 4, 4)
path.BE[3, 1:2] <- NA
path.BE[4, 3] <- NA
starting.BE <- matrix("", 4, 4)
starting.BE[3, 1:2] <- "u35"
starting.BE[4, 3] <- "u57"
BE <- simMatrix(path.BE, starting.BE)

residual.error <- diag(4)
residual.error[1,2] <- residual.error[2,1] <- NA
RPS <- symMatrix(residual.error, "n31")

loading <- matrix(0, 12, 4)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loading[7:9, 3] <- NA
loading[10:12, 4] <- NA
LY <- simMatrix(loading, "u79")

RTE <- symMatrix(diag(12))

SEM.Model <- simSetSEM(RPS = RPS, BE = BE, LY = LY, RTE = RTE)
```

The trivial model misspecification can be specified as

```
mis.path.BE <- matrix(0, 4, 4)
mis.path.BE[4, 1:2] <- NA
mis.BE <- simMatrix(mis.path.BE, "u1")

mis.loading <- matrix(NA, 12, 4)
mis.loading[is.na(loading)] <- 0
mis.LY <- simMatrix(mis.loading, "u3")

mis.error.cor <- matrix(NA, 12, 12)
diag(mis.error.cor) <- 0
mis.RTE <- symMatrix(mis.error.cor, "n1")

SEM.Mis.Model <- simMisspecSEM(BE = mis.BE, LY = mis.LY, RTE = mis.RTE)
```

Next, we need to specify the distribution of factors. Again, we will use a data distribution object to model the factor distributions. We will put only four distribution objects to represent the distribution of four factors.

```
facDist <- simDataDist(chi5, chi5, n1, n1)
```

Because Factors 1 and 2, Factor 3, and Factor 4 are in the different parts of the regression chain. In the sequential data generation, the multivariate distribution of the first two factors will be built first, which their marginal distributions are chi-squared distributed. Then, the residual from Factor 3 is generated, which is normally distributed, and is combined with the predicted score from the first two factors. Finally, the normal residual from Factor 4 is generated and is combined with the predicted score from the other factors.

The data object can be specified as

```
dataTemplate <- simData(SEM.Model, 500, SEM.Mis.Model, sequential=TRUE, facDist=facDist)
```

There are two additional arguments. The sequential argument is to use the sequential method of data generation. The `facDist` argument is to put the factor distribution objects.

The model object can be specified as

```
SimModel <- simModel(CFA.Model, estimator="mlr")
```

The `mlr` is the maximum likelihood estimator with robust Huber-White standard error with Yuan-Bentler T2 scaled test statistic.

The result object can be specified and investigated by

```
simOut <- simResult(1000, dataTemplate, modelTemplate)
getCutoff(simOut, 0.05)
plotCutoff(simOut, 0.05)
summaryParam(simOut)
```

Syntax Summary

The summary of the whole script in this example is

```
1   library(simsem)
2
3   u35 <- simUnif(0.3, 0.5)
4   u57 <- simUnif(0.5, 0.7)
5   u1 <- simUnif(-0.1, 0.1)
6   u3 <- simUnif(-0.3, 0.3)
7   n1 <- simNorm(0, 0.1)
8   n31 <- simNorm(0.3, 0.1)
9   u79 <- simUnif(0.7, 0.9)
10  chi5 <- simChisq(5)
11
12  path.BE <- matrix(0, 4, 4)
13  path.BE[3, 1:2] <- NA
14  path.BE[4, 3] <- NA
15  starting.BE <- matrix("", 4, 4)
16  starting.BE[3, 1:2] <- "u35"
17  starting.BE[4, 3] <- "u57"
18  BE <- simMatrix(path.BE, starting.BE)
19  residual.error <- diag(4)
20  residual.error[1,2] <- residual.error[2,1] <- NA
21  RPS <- symMatrix(residual.error, "n31")
22  loading <- matrix(0, 12, 4)
23  loading[1:3, 1] <- NA
24  loading[4:6, 2] <- NA
25  loading[7:9, 3] <- NA
26  loading[10:12, 4] <- NA
27  LY <- simMatrix(loading, "u79")
28  RTE <- symMatrix(diag(12))
29  SEM.Model <- simSetSEM(RPS = RPS, BE = BE, LY = LY, RTE = RTE)
30
31  mis.path.BE <- matrix(0, 4, 4)
32  mis.path.BE[4, 1:2] <- NA
33  mis.BE <- simMatrix(mis.path.BE, "u1")
34  mis.loading <- matrix(NA, 12, 4)
35  mis.loading[is.na(loading)] <- 0
36  mis.LY <- simMatrix(mis.loading, "u3")
37  mis.error.cor <- matrix(NA, 12, 12)
38  diag(mis.error.cor) <- 0
39  mis.RTE <- symMatrix(mis.error.cor, "n1")
40  SEM.Mis.Model <- simMisspecSEM(BE = mis.BE, LY = mis.LY, RTE = mis.RTE)
41
```

```
42  facDist <- simDataDist(chi5, chi5, n1, n1)
43  dataTemplate <- simData(SEM.Model, 500, SEM.Mis.Model, sequential=TRUE, facDist=facDist)
44  modelTemplate <- simModel(SEM.Model, estimator="mlr")
45  simOut <- simResult(1000, dataTemplate, modelTemplate)
46  getCutoff(simOut, 0.05)
47  plotCutoff(simOut, 0.05)
48  summaryParam(simOut)
```

Remark

1) The regression residual could be nonnormal distribution. For example, the residual distribution of the third and the fourth factors are reversed chi-squared distribution with degree of freedom of 10. The syntax in Line 42 can be changed as

```
chi10 <- simChisq(10)
facDist <- simDataDist(chi5, chi5, chi10, chi10, reverse=c(F, F, T, T))
```

2) The measurement error could be nonnormal distribution as well. For example, the measurement error are in *t*-distribution with 10 degrees of freedom. The syntax in Line 42-43 can be changed as

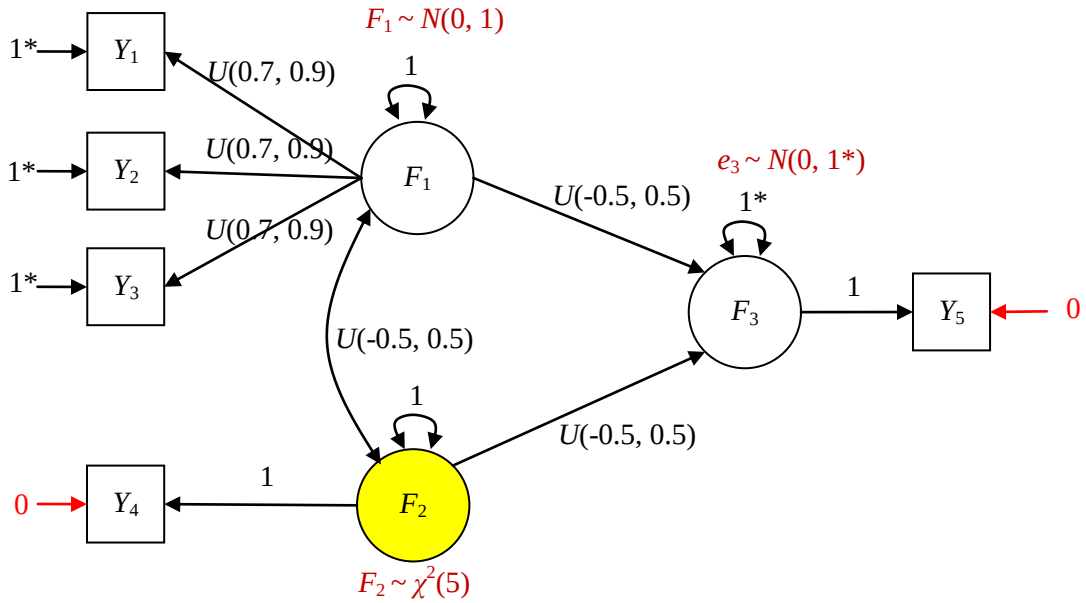
```
t10 <- simT(10)
facDist <- simDataDist(chi5, chi5, n1, n1)
errorDist <- simDataDist(t10, p=12)
dataTemplate <- simData(SEM.Model, 500, SEM.Mis.Model, sequential=TRUE, facDist=facDist,
    errorDist=errorDist)
```

The data distribution object representing the measurement error distribution is added to the `errorDist` argument of the constructor of data object.

Example 11: Single Indicator

Model Description

This example will show how to create a factor with a single indicator. We will not introduce any new syntax here; however, we will emphasize on the mean and variance vectors. Let's discuss about a model with three factors, which two of them are single-indicator factor and the other factor has three indicators. One full factor and one single-indicator factor predict the other single-indicator factor. The loading of the full factor is uniformly distribution from 0.7 to 0.9 and the error variance is set to make the indicator variance of 1. The factor loadings of the single-indicator factors are freely estimated with a parameter of 1. The factor loading is free instead of the factor variance for keeping the scale of factor variance as 1 and making the covariance matrix mean correlation matrix. The error variances of the single-indicator factors are fixed to 0. The variances of all factors are fixed to 1. The exogenous factor correlation is uniformly distributed from -0.5 to 0.5, as well as the regression paths toward the only endogenous factor. We still have model misspecification in error correlations (normally distributed with the mean of 0 and standard deviation of 1). The misspecified error correlations are not applicable to the indicators in the single-indicator factor. The exogenous single-indicator factor is chi-squared distributed with three degrees of freedom.



1* = Residual variance that makes indicator variance of 1

Trivially Misspecification:
All error correlations have $N(0, 0.1)$.

Syntax

All relevant distribution objects can be specified as

```
u79 <- simUnif(0.7, 0.9)
u5 <- simUnif(-0.5, 0.5)
n01 <- simNorm(0, 1)
c5 <- simChisq(5)
```

Factor loading can be specified as

```
loading <- matrix(0, 5, 3)
loading[1:3, 1] <- NA
loading[4, 2] <- NA
loading[5, 3] <- NA
loadingVal <- matrix(0, 5, 3)
loadingVal[1:3, 1] <- "u79"
loadingVal[4, 2] <- 1
loadingVal[5, 3] <- 1
LY <- simMatrix(loading, loadingVal)
```

Notice that the factor loadings of Indicators 4 and 5 are free and set their parameter values as 1.

The factor correlation can be specified as

```
facCor <- diag(3)
facCor[2, 1] <- NA
facCor[1, 2] <- NA
RPS <- symMatrix(facCor, "u5")
```

The regression paths among factors can be specified as

```
path <- matrix(0, 3, 3)
path[3, 1] <- NA
path[3, 2] <- NA
BE <- simMatrix(path, "u5")
```

The error correlation can be specified as

```
RTE <- symMatrix(diag(5))
```

Importantly, the indicator variance (not measurement error variance) can be specified as

```
VY <- simVector(c(NA, NA, NA, 0, 0), 1)
```

The indicator variances of the first three indicators are set as free and have parameter values of 1. It means that the error variances are free and the parameter values of the error variances are the values that make the indicator variances equal 1. The last two indicators, the single-indicator factors, are not free and fixed as 0. For this package, if the total indicator variance is set to 0, it means that error variance is set to 0. This feature is made to allow users to set measurement error of 0 while allowing them to set the total variance of other variables at the same time.

The set of SEM object can be specified as

```
SEM.Model <- simSetSEM(LY=LY, RPS=RPS, BE=BE, RTE=RTE, VY=VY)
```

The trivial model misspecification can be specified as

```
errorCorMis <- diag(5)
errorCorMis[1:3, 1:3] <- NA
errorCorMis <- diag(5)
RTE.mis <- symMatrix(errorCorMis, n01)

SEM.Model.Mis <- simMisspecSEM(RTE=RTE.mis)
```

The distribution of factors (a multiple-indicators factor and two single-indicator factors) can be specified as

```
facDist <- simDataDist(n01, c5, n01)
```

The data object, model object, and result object can be specified as

```
SimData <- simData(SEM.Model, 200, misspec=SEM.Model.Mis, sequential=TRUE, facDist=facDist)
SimModel <- simModel(SEM.Model, estimator="mlm")
Output <- simResult(1000, SimData, SimModel)
getCutoff(Output, 0.05)
plotCutoff(Output, 0.05)
summaryParam(Output)
```

Syntax Summary

The summary of the whole script in this example is

```
1   library(simsem)
2
3   u79 <- simUnif(0.7, 0.9)
4   u5 <- simUnif(-0.5, 0.5)
5   n01 <- simNorm(0, 1)
6   c5 <- simChisq(5)
7
8   loading <- matrix(0, 5, 3)
9   loading[1:3, 1] <- NA
10  loading[4, 2] <- NA
11  loading[5, 3] <- NA
12  loadingVal <- matrix(0, 5, 3)
13  loadingVal[1:3, 1] <- "u79"
14  loadingVal[4, 2] <- 1
15  loadingVal[5, 3] <- 1
16  LY <- simMatrix(loading, loadingVal)
17
18  facCor <- diag(3)
19  facCor[2, 1] <- NA
20  facCor[1, 2] <- NA
21  RPS <- symMatrix(facCor, "u5")
```

```
22
23  path <- matrix(0, 3, 3)
24  path[3, 1] <- NA
25  path[3, 2] <- NA
26  BE <- simMatrix(path, "u5")
27
28  RTE <- symMatrix(diag(5))
29
30  VY <- simVector(c(NA, NA, NA, 0, 0), 1)
31
32  SEM.Model <- simSetSEM(LY=LY, RPS=RPS, BE=BE, RTE=RTE, VY=VY)
33
34  errorCorMis <- diag(5)
35  errorCorMis[1:3, 1:3] <- NA
36  errorCorMis <- diag(5)
37  RTE.mis <- symMatrix(errorCorMis, n01)
38
39  SEM.Model.Mis <- simMisspecSEM(RTE=RTE.mis)
40
41  facDist <- simDataDist(n01, c5, n01)
42
43  SimData <- simData(SEM.Model, 200, misspec=SEM.Model.Mis, sequential=TRUE, facDist=facDist)
44  SimModel <- simModel(SEM.Model, estimator="mlm")
45  Output <- simResult(1000, SimData, SimModel)
46  getCutoff(Output, 0.05)
47  plotCutoff(Output, 0.05)
48  summaryParam(Output)
```

Remark

1) The example makes the single-indicator factors by freeing the factor loadings and specifying the indicator intercepts to specify the mean. Instead, we can specify the factor variance and factor mean as another parameterization of the single indicator. The syntax is more complicated. The factor loading in Lines 8-16 can be changed to

```
loading <- matrix(0, 5, 3)
loading[1:3, 1] <- NA
loading[4, 2] <- 1
loading[5, 3] <- 1
loadingVal <- matrix(0, 5, 3)
loadingVal[1:3, 1] <- "u79"
LY <- simMatrix(loading, loadingVal)
```

The factor loadings of the single-indicator factors are fixed to 1. In addition, Lines 31-32 can be changed to

```
VE <- simVector(c(1, NA, NA), c(0, 1, 1))
ME <- simVector(c(0, NA, NA), c(0, 0, 0))
TY <- simVector(c(NA, NA, NA, 0, 0), rep(0, 5))
SEM.Model <- simSetSEM(LY=LY, RPS=RPS, BE=BE, RTE=RTE, VY=VY, VE=VE, ME=ME, TY=TY)
```

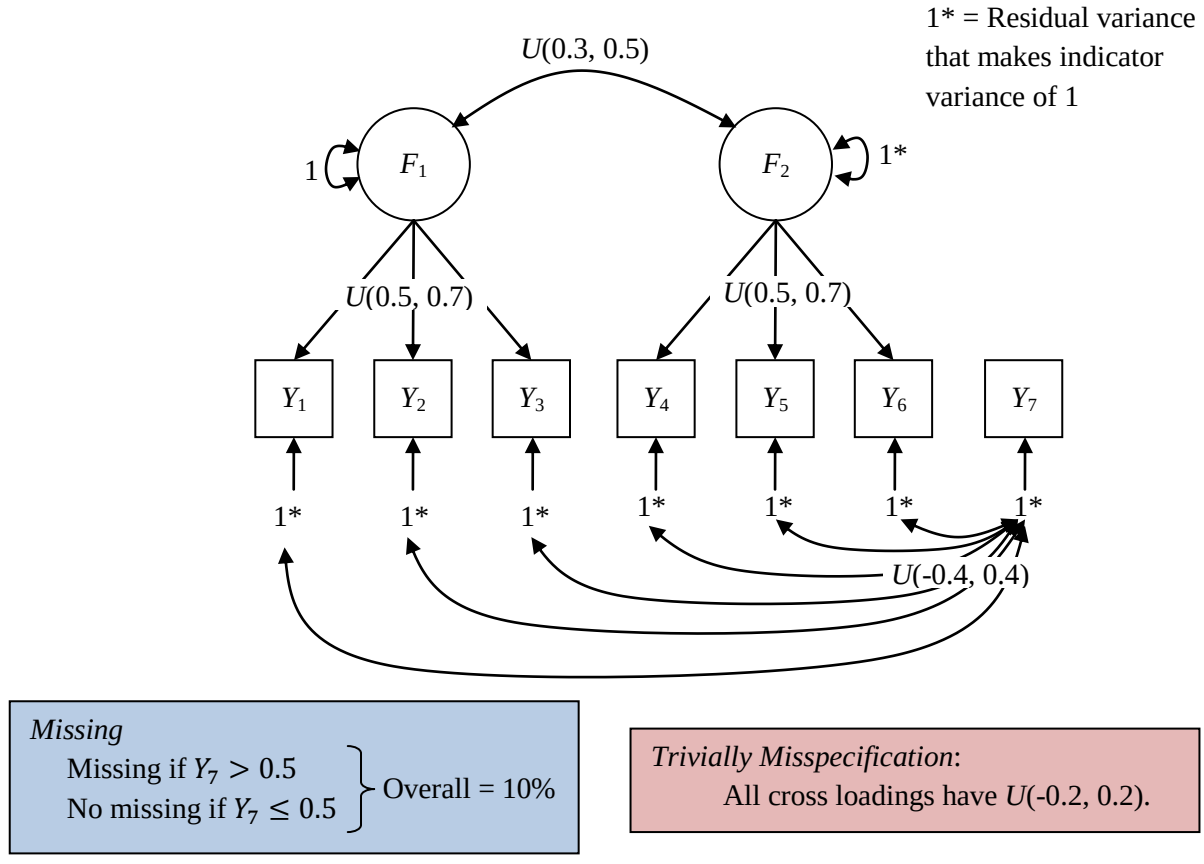
`VE` is the total variance of factors. Here we fix the variance of the first factor as 1. The variances of the single-indicator factors are free and their parameter values are 1. `ME` is the total mean of factors. The overall mean of the first factor is fixed to 1. The overall mean of the second and the third factors are free and their parameter values are 1. `TY` is the measurement intercept. The first three indicators intercepts are freely estimates and their parameter values are 1. The measurement intercepts of the single-factor indicator are fixed to 1. The measurement intercepts of the other two indicators are fixed to 1.

Example 12: Missing at Random and Auxiliary Variable

Model Description

This example will also show how to model an auxiliary variable in the data and how to create the data with missing at random given the auxiliary variable. The auxiliary variable is not the variables of interest but it influences the missing pattern (e.g., the higher the value of the auxiliary variable is, the higher the chance of missing in a target variable). In this example, we will make all target variables have a chance of missing if the specified auxiliary variables are greater a given value (e.g., its mean).

The model in this example is two-factor confirmatory factor analysis model with three indicators each. The factor loadings are uniformly distributed from 0.5 to 0.7. The factor correlation is uniformly distributed from 0.3 to 0.5. The error covariances are set to make the indicator variance of 1. Next, an auxiliary variable with the variance of 1 is included in the model and correlates with measurement errors range from -0.4 to 0.4 in uniform distribution. For the model misspecification, the cross loadings are ranged from -0.2 to 0.2 in uniform distribution. The overall amount of missing values is 10%. However, the data is missing if and only if the auxiliary-variable value is greater than 0.5. Because the auxiliary variable has the mean of 0 and the variance of 1, the auxiliary-variable value can be interpreted as standard score.



Syntax

All relevant distribution objects can be specified as

```
u57 <- simUnif(0.5, 0.7)
u4 <- simUnif(-0.4, 0.4)
u35 <- simUnif(0.3, 0.5)
```

The parameter model can be specified as

```
loading <- matrix(0, 7, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
LX <- simMatrix(loading, "u57")

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, "u35")

error.cor <- diag(7)
error.cor[1:6, 7] <- NA
error.cor[7, 1:6] <- NA
RTD <- symMatrix(error.cor, "u4")

VX <- simVector(rep(NA, 7), 1)

CFA.Model.Aux <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD, VX = VX)
```

There are two interesting things here. First, the correlations between the auxiliary variable and the target-variables measurement errors are free. Second, the `VX` command is the total variances of indicators. As shown in Example 1, this is the default of the program. This command is explicitly shown here to emphasize the existence of the auxiliary variable.

The trivial model misspecification can be specified as

```
mis.loading <- matrix(0, 7, 2)
mis.loading[1:3, 2] <- NA
mis.loading[4:6, 1] <- NA
mis.LY <- simMatrix(mis.loading, "u2")

CFA.Mis.Model <- simMisspecCFA(LY = mis.LY)
```

Notice that the auxiliary variables are not involved with trivial model misspecification: the cross loadings are not related to Variable 7.

We can create a data object (with 200 cases) based on this model by

```
SimData <- simData(CFA.Model.Aux, 200, misspec = CFA.Mis.Model)
```

Actually, we can use `CFA.Model.Aux` for creating a model object. However, accounting for an auxiliary variable is a tedious task. This package allows you to only set the model for the target model and specify the list of auxiliary variables. Then, the package will create the model accounting for auxiliary variables (in the full information maximum likelihood) or take out the list of auxiliary variables and analyze by the target model in the multiple imputation.

Now, you may specify the target model (i.e., with six variables). If you already have the model with auxiliary variables, you may extract only a part of the model, `CFA.Model.Aux`, by the `extract` function as

```
CFA.Model <- extract(CFA.Model.Aux, y=1:6)
```

The y argument is the index of target variables. We need to keep the first six variables for the target model. You may use the summary function to check the new object.

The model object can be created by

```
SimModel <- simModel(CFA.Model)
```

This model object will have only six target variables. Then, we need to specify the missing object to impose missing values that the occurrence of missing values depends on the auxiliary variable. To repeat, the missing values are imposed only when the covariate value of a case is over 0.5. The missing object can be specified as

```
SimMissing <- simMissing(pmMAR=0.1, cov=7, numImps=5, threshold=0.5)
```

The pmMAR argument is the percentage of overall missing value. The cov argument is the index of covariate in a dataset. The numImps argument is the number of imputations, which implies that the following data analysis will use multiple imputation. The pCov argument is the proportion of data that have missing values based on MAR process.

The selection is based on the values of the specified covariate. The proportion of the auxiliary variable over 0.5 is approximately 31%. Therefore, there is a 32% (overall missing / proportion of auxiliary variable = .1 / .31 = .32) chance that the other variables corresponding to the case which the value of auxiliary variable is over 0.5 will be missing.

Let's generate a dataset, impose missing values, and analyze the generated data. The syntax is

```
data <- run(SimData, dataOnly=F)
data <- run(SimMissing, data)
out <- run(SimModel, data, simMissing=SimMissing)
summary(out)
```

The first line is to create data. The dataOnly = F means to not just provide the data only but provide parameter values and other setups in the data generation as well. The second line is to impose missing values into the data. The third line is to analyze data with a specified missing object. The list of auxiliary variables is the cov argument in the missing object. The fourth line is to summarize the output from data analysis. In multiple imputation, all variables are used in the multiple imputation and the auxiliary variables will be excluded from the data analysis. If full information maximum likelihood is used (not specify the numImps argument in the missing object), the auxiliary variables will be included in the model. See Remark for further details.

The result object can be specified and investigated by

```
Output <- simResult(1000, SimData, SimModel, SimMissing)
getCutoff(Output, 0.05)
plotCutoff(Output, 0.05)
summaryParam(Output)
```

Syntax Summary

The summary of the whole script in this example is

```
1  library(simsem)
2
3  u57 <- simUnif(0.5, 0.7)
4  u4 <- simUnif(-0.4, 0.4)
```

```
5   u35 <- simUnif(0.3, 0.5)
6   u2 <- simUnif(-0.2, 0.2)
7   n01 <- simNorm(0, 1)
8
9   loading <- matrix(0, 7, 2)
10  loading[1:3, 1] <- NA
11  loading[4:6, 2] <- NA
12  LX <- simMatrix(loading, "u57")
13  latent.cor <- matrix(NA, 2, 2)
14  diag(latent.cor) <- 1
15  RPH <- symMatrix(latent.cor, "u35")
16  error.cor <- diag(7)
17  error.cor[1:6, 7] <- NA
18  error.cor[7, 1:6] <- NA
19  RTD <- symMatrix(error.cor, "u4")
20  VX <- simVector(rep(NA, 7), 1)
21  CFA.Model.Aux <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD, VX = VX)
22
23  mis.loading <- matrix(0, 7, 2)
24  mis.loading[1:3, 2] <- NA
25  mis.loading[4:6, 1] <- NA
26  mis.LY <- simMatrix(mis.loading, "u2")
27  CFA.Mis.Model <- simMisspecCFA(LY = mis.LY)
28
29  SimData <- simData(CFA.Model.Aux, 200, misspec = CFA.Mis.Model)
30
31  CFA.Model <- extract(CFA.Model.Aux, y=1:6)
32
33  SimMissing <- simMissing(pmMAR=0.1, cov=7, numImps=5, threshold=0.5)
34
35  SimModel <- simModel(CFA.Model)
36
37  Output <- simResult(1000, SimData, SimModel, SimMissing)
38  getCutoff(Output, 0.05)
39  plotCutoff(Output, 0.05)
40  summaryParam(Output)
```

Remark

1) To use the full-information maximum likelihood, the syntax in Line 33 can be specified as

```
SimMissing <- simMissing(pmMAR=0.1, cov=7, threshold=0.5)
```

   The `numImps` is not specified here (you may set it as 0).

2) The techniques used to account for auxiliary variables in the full information maximum likelihood method in this package are both extra-dependent-variables and saturated-correlates approaches. For all path analysis models, the extra dependent-variables approach is used. For all confirmatory factor analysis and structural equation modeling models, the saturated-correlates approach is mainly used. When an indicator with no measurement error exists (see Examples 11 or 12), the extra-dependent-variables approach is used such that the factors on the indicators predict the auxiliary variables. See here for the details of both approaches.

3) The list of auxiliary variables can be specified when building the model object. Line 40 can be changed to

```
SimModel <- simModel(CFA.Model, auxiliary = 7)
```

   The auxiliary argument is to put the index or the variable name of the desired auxiliary variable.

4) If we want to create data that the missingness depends on the value of an auxiliary variable but analyze the data by not accounting for the auxiliary variable, the syntax in Lines 33-35 can be changed to

```
SimMissing <- simMissing(pmMAR=0.1, cov=7, numImps=5, threshold = 0.5, covAsAux=FALSE)

SimModel <- simModel(CFA.Model, indLab=1:6)
```

The covAsAux argument in the simMissing function means to not include the specified covariate as auxiliary variable but treat as the real variable in data analysis. In other words, all seven variables are used in data analysis. Therefore, the indLab argument in the simModel is used to extract only the first six variables in data analysis. By this way, the auxiliary variable will not be accounted, as well as analyzed, by model object.

Functions Review

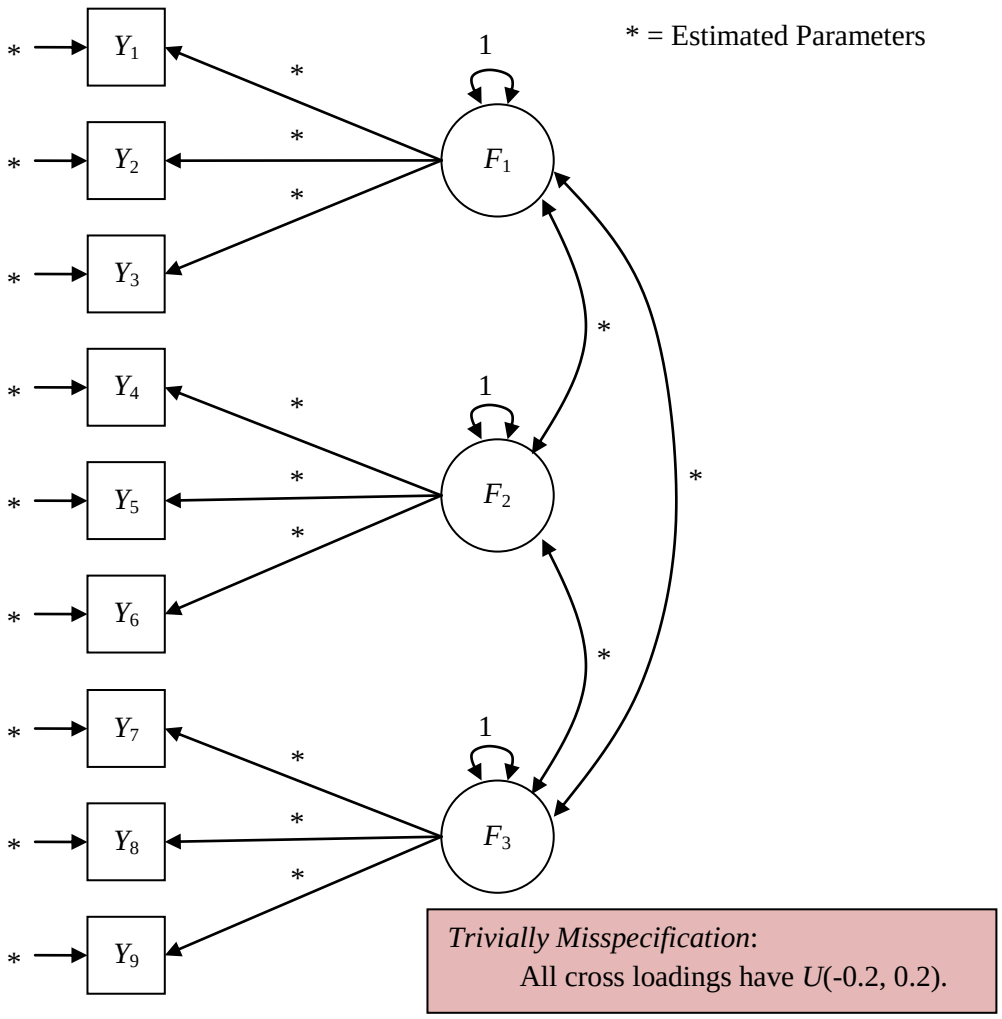
Functions	Usage
extract	Extract a subset of a specified object

Example 13: Analyzing Real Data

Model Description

This example will show how to analyze data by this package. This package does not truly aim for data analysis, which lavaan and OpenMx have already taken care of. This package, rather, will use the result for data analysis as parameters in data generation of a simulation study. This procedure will yield a distribution of fit indices from simulated data, which can compare with the result of the real data. This procedure is called a Monte Carlo approach for model evaluation.

This example will show how to analyze the Holzinger and Swineford (1939) data that have three factors with three indicators each. The data will be analyzed by confirmatory factor analysis with three factors with three indicators each. The fixed factor method of scale identification is used. The resulting standardized parameter estimates will be used for data generation. Then, cross-loadings with uniform distribution from -0.2 to 0.2 will be put on top of the parameter estimates to generate trivial model misspecification. Then, the generated data (from both real parameters and trivial misspecification) will be analyzed and provide the distribution of fit indices.



Syntax

In this example, we do not need to specify parameter values. Therefore, only parameter specifications are needed. Thus, the matrix object, symmetric matrix object, vector object, and the specification using `simSet` functions are not needed. Instead, the parameter specifications can be used and combined together by `simParam` functions (`simParamCFA`, `simParamPath`, or `simParamSEM`). For the CFA model, the `simParamCFA` is needed:

```
loading <- matrix(0, 9, 3)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loading[7:9, 3] <- NA
model <- simParamCFA(LY=loading)
```

The `loading` matrix shows which elements are freely estimated and which elements are fixed to 0. The `simParamCFA` function will build the analysis model. By default of this function, all factor variances are fixed to 1, all factor covariances are free, all error variances are free, all error covariances are fixed to 0, all factor means are fixed to 0, and all measurement intercepts are free. See the Remark section for the manual specification of all matrices. The Holzinger and Swineford (1939) data can be found from the `lavaan` package by

```
library(lavaan)
summary(HolzingerSwineford1939)
```

The target variables used for the analysis are x1 to x9. Thus, the model object can be specified by

```
SimModel <- simModel(model, indLab=paste("x", 1:9, sep=""))
```

The indLab argument is used to select a subset of variables for data analysis. The paste("x", 1:9, sep="")) is used to get a vector of x1, x2, …, x9. The model object will search for these variable names and select for data analysis. The order of the names must be the same as the order of indicators in the analysis model. The data can be analyzed by

```
out <- run(SimModel, HolzingerSwineford1939)
summary(out)
```

The data and the model object can also be used to build a simulation study based on the parameter estimates obtained from the output. The default of this package is to use standardized parameter estimates so that the trivial model misspecification can be added in a meaningful way (note that some models are not such as growth curve model). The simulation study can be specified by the runFit function:

```
output <- runFit(SimModel, HolzingerSwineford1939, 1000)
```

The result of the runFit function is a result object. The fit index cutoff using the alpha level of .05 can be visualized by the plotCutoff function:

```
plotCutoff(output, 0.05)
```

Because we have the analysis output from the real data, we can compare the fit indices obtained by the real data analysis and the fit indices from the simulation study by the pValue function:

```
pValue(out, output)
```

The first argument of the pValue function is the analysis output from the real data. The second argument is the result object from the simulation study. The usedFit argument can be used to select desired fit indices. The result provides two extra values: andRule and orRule. The andRule is based on the principle that the model is retained only when all fit indices provide good fit. The proportion is calculated from the number of replications that have all fit indices indicating a better model than the observed data. The proportion from the andRule is the most stringent rule in retaining a hypothesized model. The orRule is based on the principle that the model is retained only when at least one fit index provides good fit. The proportion is calculated from the number of replications that have at least one fit index indicating a better model than the observed data. The proportion from the orRule is the most lenient rule in retaining a hypothesized model. Both rules are based on only selected fit indices. The default uses seven fit indices: Chi-square, AIC, BIC, RMSEA, CFI, TLI, and SRMR.

The previous code does not account for trivial model misspecification yet. The simulated data are generated from the real result only. The trivial model misspecification can be added by creating the misspecification object first and putting the object in the misspec argument of the simFit function:

```
u2 <- simUnif(-0.2, 0.2)
loading.mis <- matrix(NA, 9, 3)
loading.mis[is.na(loading)] <- 0
LY.mis <- simMatrix(loading.mis, "u2")
misspec <- simMisspecCFA(LY=LY.mis)
output2 <- runFit(SimModel, HolzingerSwineford1939, 1000, misspec=misspec)
pValue(out, output2)
```

From here, the pValue from the model with trivial misspecification will be larger than the model without trivial misspecification. Therefore, the chance to retain the hypothesized model is higher when accounting for trivial model misspecification.

Syntax Summary

The summary of the whole script in this example is

```
1  library(simsem)
2  library(lavaan)
3  loading <- matrix(0, 9, 3)
4  loading[1:3, 1] <- NA
5  loading[4:6, 2] <- NA
6  loading[7:9, 3] <- NA
7  model <- simParamCFA(LY=loading)
8  SimModel <- simModel(model, indLab=paste("x", 1:9, sep=""))
9  out <- run(SimModel, HolzingerSwineford1939)
10
11 ### Making result object without trivial model misspecification
12 #output <- runFit(SimModel, HolzingerSwineford1939, 1000)
13 #pValue(out, output)
14
15 u2 <- simUnif(-0.2, 0.2)
16 loading.mis <- matrix(NA, 9, 3)
17 loading.mis[is.na(loading)] <- 0
18 LY.mis <- simMatrix(loading.mis, "u2")
19 misspec <- simMisspecCFA(LY=LY.mis)
20 output2 <- runFit(SimModel, HolzingerSwineford1939, 1000, misspec=misspec)
21 pValue(out, output2)
```

Remark

1) All parameters can be explicitly specified in the simParamCFA function by changing Line 7 as

```
facCov <- matrix(NA, 3, 3)
diag(facCov) <- 1
errorCov <- diag(NA, 9)
intercept <- rep(NA, 9)
facMean <- rep(0, 3)
model <- simParamCFA(LY=loading, PS=facCov, TE=errorCov, TY=intercept, AL=facMean)
```

2) The simulated data can be generated by unstandardized parameters. The usedStd argument in the runFit function can be specified as FALSE to use standardized parameters for data generation. The Line 20 can be changed to

```
output2 <- runFit(SimModel, HolzingerSwineford1939, 1000, misspec=misspec, usedStd=FALSE)
```

3) Most of arguments in the runFit function are similar to the simData function or the simResult function. The help page can be found by

```
method?runFit
```

The useful arguments are sequential (for the sequential method for data generation) and multicore (for parallelization across processors).

4) The usedFit argument in the pValue function can be specified so the function will find and calculate all *p* values based on only interested fit indices. For example, if we are interested in only RMSEA and CFI, Line 21 can be changed as

```
pValue(out, output2, usedFit=c("RMSEA", "CFI"))
```

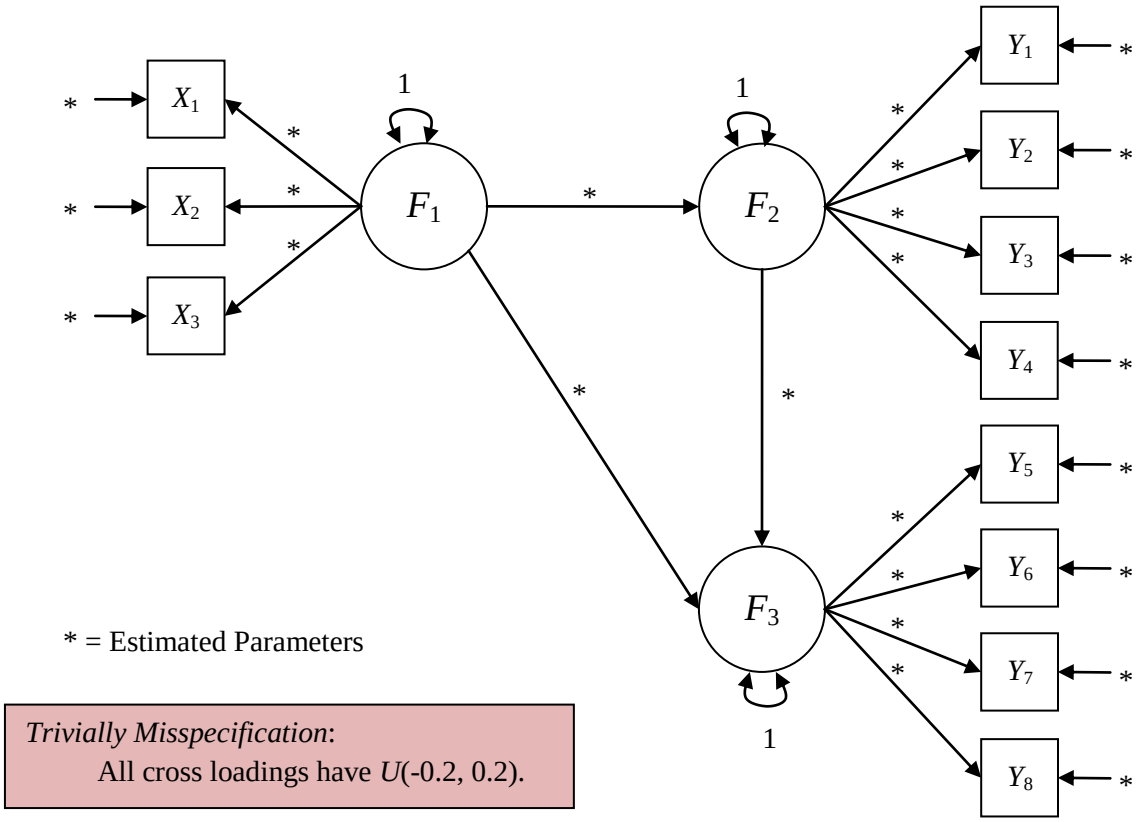
Functions Review

Functions	Usage
simParamCFA	Create a set of free parameters for CFA
simParamPath	Create a set of free parameters for path analysis
simParamSEM	Create a set of free parameters for SEM
runFit	Use the result from real data to run a Monte Carlo simulation
pValue	Find a *p* value

Example 14: Analyzing Real Data with Multiple Imputation

Model Description

This example will show how to analyze data by this package using multiple imputation and show how to use a Monte Carlo approach for model evaluation. The PoliticalDemocracy from Bollen (1989; data are available in the lavaan package) will be used. The analysis model is a mediation model among three factors. The fixed factor method of scale identification is used. The resulting standardized parameter estimates will be used for data generation. Then, cross-loadings with uniform distribution from -0.2 to 0.2 will be put on top of the parameter estimates to generate trivial model misspecification. Then, the generated data (from both real parameters and trivial misspecification) will be analyzed and provide the distribution of fit indices.



Syntax

The factor loading parameters can be specified as

```
loading <- matrix(0, 11, 3)
loading[1:3, 1] <- NA
loading[4:7, 2] <- NA
loading[8:11, 3] <- NA
```

The regression among factors can be specified as

```
path <- matrix(0, 3, 3)
path[2:3, 1] <- NA
path[3, 2] <- NA
```

Because this model is SEM, the simParamSEM is needed to build the model of free parameters. This function requires only factor loading matrix and regression coefficient among factors:

```
param <- simParamSEM(LY=loading, BE=path)
```

By default of this function, all factor (residual) variances are fixed to 1, all exogenous factor covariances are free, all endogenous factor covariances are fixed to 0, all error variances are free, all error covariances are fixed to 0, all factor means (intercept) are fixed to 0, and all measurement intercepts are free. See the Remark section for the manual specification of all matrices. The PoliticalDemocracy can be found from the lavaan package by

```
library(lavaan)
summary(PoliticalDemocracy)
```

To show how to analyze real data by multiple imputation, we need to make this data missing. We will use the imposeMissing function to make this data missing completely at random by 3% as

```
usedData <- imposeMissing(PoliticalDemocracy, pmMCAR=0.03)
```

The imposeMissing function is exactly the same as making a missing object and run it. The imposeMissing function is built as a shortcut for imposing missing data. Now, pretend that the usedData is the actual data that we obtained.

The target variables used for the analysis are x1 to x3 and y1 to y8. Thus, the model object can be specified by

```
model <- simModel(param, indLab=c(paste("x", 1:3, sep=""), paste("y", 1:8, sep="")))
```

The data can be analyzed by

```
miss <- simMissing(numImps=5)
out <- run(model, usedData, miss)
summary(out)
```

Because we need to use multiple imputation, the missing object is required. The numImps argument is the number of imputed data. The miss object is put in the run function to analyze data by multiple imputation.

The trivial model misspecification with cross-loadings ranged from -0.2 to 0.2 can be specified as

```
u2 <- simUnif(-0.2, 0.2)
loading.mis <- matrix(NA, 11, 3)
loading.mis[is.na(loading)] <- 0
LY.mis <- simMatrix(loading.mis, "u2")
misspec <- simMisspecSEM(LY=LY.mis)
```

The simulation study based on the real result with trivial model misspecification can be specified by the runFit function and the pValue function can be used to find the *p* value based on the Monte Carlo approach as

```
output <- runFit(model, usedData, 200, misspec=misspec, missModel=miss)
pValue(out, output)
```

Note that the runFit function will use the same missing data pattern on the real data imposing to all simulated data. See the Remark section for imposing different patterns.

Syntax Summary

The summary of the whole script in this example is

```
1   library(simsem)
2   library(lavaan)
3   loading <- matrix(0, 11, 3)
4   loading[1:3, 1] <- NA
5   loading[4:7, 2] <- NA
6   loading[8:11, 3] <- NA
7   path <- matrix(0, 3, 3)
8   path[2:3, 1] <- NA
9   path[3, 2] <- NA
10  param <- simParamSEM(LY=loading, BE=path)
11
12  usedData <- imposeMissing(PoliticalDemocracy, pmMCAR=0.03)
13
14  model <- simModel(param, indLab=c(paste("x", 1:3, sep=""), paste("y", 1:8, sep="")))
15  miss <- simMissing(numImps=5)
16  out <- run(model, usedData, miss)
17
18  u2 <- simUnif(-0.2, 0.2)
19  loading.mis <- matrix(NA, 11, 3)
20  loading.mis[is.na(loading)] <- 0
21  LY.mis <- simMatrix(loading.mis, "u2")
22  misspec <- simMisspecSEM(LY=LY.mis)
23  output <- runFit(model, usedData, 200, misspec=misspec, missModel=miss)
24  pValue(out, output)
```

Remark

1) All parameters can be explicitly specified in the simParamSEM function by changing Line 10 as

```
param <- simParamSEM(LY=loading, BE=path, PS=diag(3), TE=diag(NA, 11), AL=rep(1, 3),
     TY=rep(NA, 11))
```

2) The missing data can be handled by full information maximum likelihood (instead of multiple imputation) by changing Line 23 as

```
output <- runFit(model, usedData, 200, misspec=misspec)
```

   The default of the runFit function is to use the full information maximum likelihood method.

3) If we want to use missing object to generate different patterns of missing data instead of using the missing pattern from the real data, we need to build a missing object and change the empiricalMissing argument in the runFit function. For example, we want to put 20% missing completely at random. First, Line 15 can be changed to

```
miss <- simMissing(pmMCAR=.20, numImps=5)
```

Next, Line 23 can be changed by specifying the `empiricalMissing` argument as `FALSE` to generate the missing pattern by the missing object:

```
output <- runFit(model, usedData, 200, misspec=misspec, empiricalMissing=FALSE, missModel=miss)
```

If you need a complete data, a blank missing object can be specified and put in the runFit runction. The blank missing object can be specified as

```
miss <- simMissing()
```

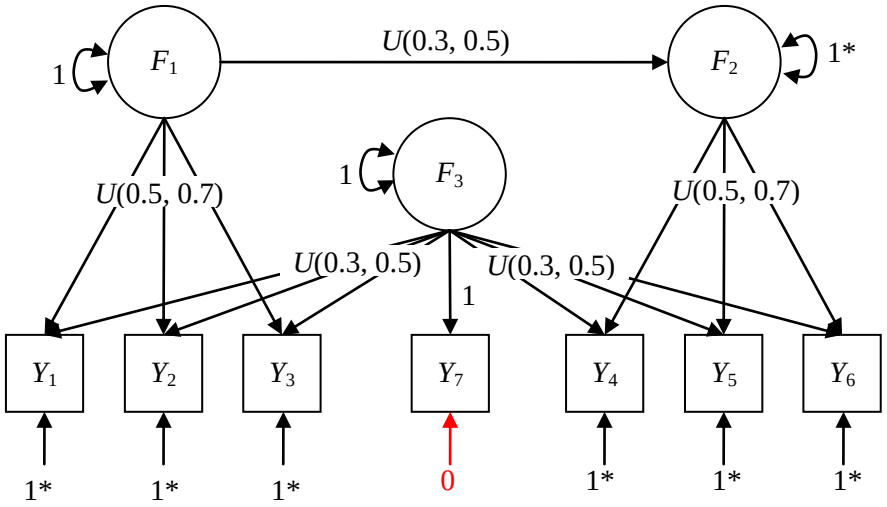
Functions Review

Functions	Usage
`imposeMissing`	Impose missing data onto a real data

Example 15: Modeling Covariate

Model Description

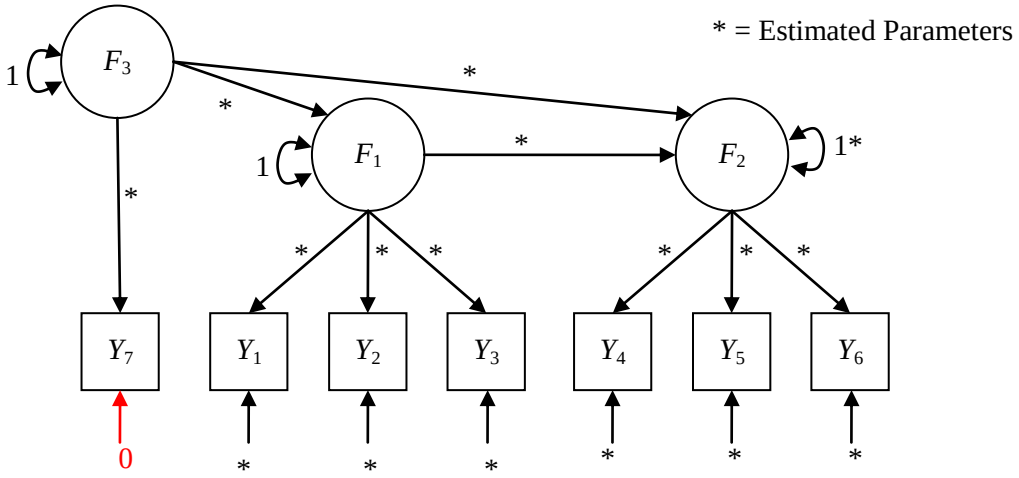
This example will show how to create a model with covariate and analyze data with or without covariate. The target model has two factors with three indicators each. The second factor is regressed on the first factor. The parameters are shown in the figure below. The variable Y_7 is an indicator-level covariate. The effect of the covariate ranges from 0.3 to 0.5 in standardized metric. The trivial misspecification is also added during the data generation process as shown in the box below.



1* = Residual variance that makes indicator variance of 1

Trivially Misspecification:
All error correlations have $N(0, 0.1)$.

We will analyze the simulated data by 1) excluding the covariate from the analysis, 2) accounting for the covariate as the model described above, 3) accounting for the covariate by orthogonalization, and 4) accounting for the covariate at the factor level, as shown in the figure below.



Syntax

All relevant distribution objects can be specified as

```
u35 <- simUnif(0.3, 0.5)
u57 <- simUnif(0.5, 0.7)
n01 <- simNorm(0, 1)
```

Factor loading can be specified as

```
loading <- matrix(0, 7, 3)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loading[1:7, 3] <- NA
loadingVal <- matrix(0, 7, 3)
loadingVal[1:3, 1] <- "u57"
loadingVal[4:6, 2] <- "u57"
loadingVal[1:6, 3] <- "u35"
loadingVal[7, 3] <- 1
LY <- simMatrix(loading, loadingVal)
```

Notice that the factor loading of Indicators 7 on Factor 3 is free with the parameter value of 1. The factor correlation can be specified as

```
RPS <- symMatrix(diag(3))
```

The regression paths among factors can be specified as

```
path <- matrix(0, 3, 3)
path[2, 1] <- NA
BE <- simMatrix(path, "u35")
```

The error correlation can be specified as

```
RTE <- symMatrix(diag(7))
```

Importantly, the indicator variance (not measurement error variance) can be specified as

```
VY <- simVector(c(rep(NA, 6), 0), rep(1, 7))
```

Similar to Example 11, the indicator variances of the first six indicators are set as free and have parameter values of 1. It means that the error variances are free and the parameter values of the error variances are the values that make the indicator variances equal 1. The last indicator, the covariate is fixed as 0. For this package, if the total indicator variance is set to 0, it means that error variance is set to 0.

This feature is made to allow users to set measurement error of 0 while allowing them to set the total variance of other variables at the same time.

The set of SEM object can be specified as

```
Cov.Model <- simSetSEM(LY=LY, RPS=RPS, BE=BE, RTE=RTE, VY=VY)
```

The trivial model misspecification can be specified as

```
errorCorMis <- diag(7)
errorCorMis[1:6, 1:6] <- NA
errorCorMis <- diag(7)
RTE.mis <- symMatrix(errorCorMis, n01)
Cov.Model.Mis <- simMisspecSEM(RTE=RTE.mis)
```

The data object can be specified as

```
SimData <- simData(Cov.Model, 200, misspec=Cov.Model.Mis)
```

The first analysis model is the model that excludes the covariate. This analysis model can be specified as

```
No.Cov.Model <- extract(Cov.Model, y=1:6, e=1:2)
model1 <- simModel(No.Cov.Model, indLab=paste("y", 1:6, sep=""))
Output1 <- simResult(100, SimData, model1)
```

The second analysis model is the model using for data generation. This analysis model can be specified as

```
model2 <- simModel(Cov.Model)
Output2 <- simResult(100, SimData, model2)
```

Before building the third analysis model, we need to know how to orthogonalize data and how to transform the data during within the result object. First, the function used for orthogonalization is `residualCovariate`. For example, if we wish to orthogonalize the Variables 2-7 by the Variable 1 in the `attitude` dataset, the function can be specified as

```
head(attitude)
dat <- residualCovariate(attitude, targetVar=2:7, covVar=1)
head(dat)
```

The first argument of the `residualCovariate` function is the target dataset. The second argument, `targetVar`, is the variables for orthogonalization. The third argument, `covVar`, is the covariate. Note that covariate can be more than one variable. The head function is to view only a first few rows of a dataset. You will notice that the second to seventh variables have been orthogonalized.

Next, we will introduce a function object that will hold all specifications of a function and can be used for analysis later. For example, if we would like to hold a specification that we wish to use the `residualCovariate` function that the target variables are Variables 2-7 and the covariate is Variable 1. The function object can be built be the `simFunction` function:

```
fun <- simFunction(residualCovariate, targetVar=2:7, covVar=1)
```

This function object can be run on the target data as

```
dat <- run(fun, attitude)
head(dat)
```

When running the function object, the first following argument can be anything that we usually put as the first argument when we call the function. For the residualCovariate function, a dataset should follow the function object name.

As described above, the function object will save a specification of a function and will used for data transformation when the package builds a result object. Thus, the third analysis model, the model using orthogonalization, can be specified as

```
ortho <- simFunction(residualCovariate, targetVar=1:6, covVar=7)
model3 <- model1
Output3 <- simResult(100, SimData, model3, objFunction=ortho)
```

The analysis model is exactly the same as the first analysis model.

The fourth analysis model that accounts the covariate in the factor level can be specified as

```
loading <- matrix(0, 7, 3)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loading[7, 3] <- NA
path <- matrix(0, 3, 3)
path[2, 1] <- NA
path[1, 3] <- NA
path[2, 3] <- NA
errorCov <- diag(NA, 7)
errorCov[7, 7] <- 0
facCov <- diag(3)
Fac.Cov.Model <- simParamSEM(LY=loading, BE=path, TE=errorCov, PS=facCov)
model4 <- simModel(Fac.Cov.Model)
Output4 <- simResult(100, SimData, model4)
```

You will notice that the results from the second and the third analysis models are similar.

If we summarize the result objects from the Analysis Model 1, 3, and 4 (which are Output1, Output3, and Output4), we will find the note that the population underlying the data generation model does not show up and we cannot find any biases in parameter estimates or standard errors. The reason is that the population underlying the data generation and the analysis model are not the same (e.g., seven indicators in data generation but six indicators in analysis model). We can still view population underlying the data generation process by the summaryPopulation function:

```
summaryPopulation(Output1)
```

We can also extract the population model underlying each replication by the getPopulation function as

```
param <- getPopulation(Output1)
```

You can see that the population underlying the data generation model includes the covariate. If we exclude the covariate from the population set, the population underlying the data generation model should be able to compare with the results from the simulation study. That is, we will exclude the population model that involves with Indicator 7 and Factor 3 and put it back into the result object. The modification of parameter model can be done by the extract function as

```
param <- extract(param, y=1:6, e=1:2)
```

The y argument is the index of indictors to be kept. The e, which is stand for eta, is the index of factors to be kept. This modified parameter values can be put back in the result object by the setPopulation function as

```
Output1 <- setPopulation(Output1, param)
summary(Output1)
```

The first argument of the setPopulation function is the target result object. The second argument is the parameter values to be put. We will notice that the summary function will provide the bias in parameter estimates and standard errors now. This procedure can be implemented to the result object from the third analysis model:

```
param <- getPopulation(Output3)
param <- extract(param, y=1:6, e=1:2)
Output3 <- setPopulation(Output3, param)
```

The same procedure cannot implement on the fourth analysis model because the analysis model is not nested in the data generation model (i.e., we have extra parameters in controlling the covariate in the factor level). We can put the appropriate parameter model instead:

```
loadingVal <- matrix(0, 7, 3)
loadingVal[1:3, 1] <- 0.6
loadingVal[4:6, 2] <- 0.6
loadingVal[7, 3] <- 1
LY <- simMatrix(loading, loadingVal)
pathVal <- matrix(0, 3, 3)
pathVal[2, 1] <- 0.4
pathVal[1, 3] <- 0.4
pathVal[2, 3] <- 0.4
BE <- simMatrix(path, pathVal)
PS <- symMatrix(facCov)
errorCovVal <- diag(0.64, 7)
errorCovVal[7, 7] <- 0
TE <- symMatrix(errorCov, errorCovVal)
Fac.Cov.Model.Full <- simSetSEM(LY=LY, PS=PS, BE=BE, TE=TE)
Output4 <- setPopulation(Output4, Fac.Cov.Model.Full)
```

That is, we put the set of matrices object as the second argument. Then the appropriate population model is put in the result object.

Syntax Summary

The summary of the whole script in this example is

```
1   library(simsem)
2   u35 <- simUnif(0.1, 0.3)
3   u57 <- simUnif(0.5, 0.7)
4   u2 <- simUnif(-0.2, 0.2)
5
6   loading <- matrix(0, 7, 3)
7   loading[1:3, 1] <- NA
8   loading[4:6, 2] <- NA
9   loading[1:7, 3] <- NA
10  loadingVal <- matrix(0, 7, 3)
11  loadingVal[1:3, 1] <- "u57"
12  loadingVal[4:6, 2] <- "u57"
13  loadingVal[1:6, 3] <- "u35"
14  loadingVal[7, 3] <- 1
15  LY <- simMatrix(loading, loadingVal)
16
17  RPS <- symMatrix(diag(3))
18
19  path <- matrix(0, 3, 3)
20  path[2, 1] <- NA
21  BE <- simMatrix(path, "u35")
22
23  RTE <- symMatrix(diag(7))
24
25  VY <- simVector(c(rep(NA, 6), 0), rep(1, 7))
26
```

```
Cov.Model <- simSetSEM(LY=LY, RPS=RPS, BE=BE, RTE=RTE, VY=VY)

loading.mis <- matrix(NA, 7, 3)
loading.mis[is.na(loading)] <- 0
loading.mis[,3] <- 0
loading.mis[7,] <- 0
LY.mis <- simMatrix(loading.mis, "u2")
misspec <- simMisspecSEM(LY=LY.mis)

SimData <- simData(Cov.Model, 200, misspec=misspec)

# First analysis model: Model without covariate
No.Cov.Model <- extract(Cov.Model, y=1:6, e=1:2)
model1 <- simModel(No.Cov.Model, indLab=paste("y", 1:6, sep=""))
Output1 <- simResult(100, SimData, model1)
param <- getPopulation(Output1)
param <- extract(param, y=1:6, e=1:2)
Output1 <- setPopulation(Output1, param)
summary(Output1)

# Second analysis model: Model accounting for covariate in the indicator level
model2 <- simModel(Cov.Model)
Output2 <- simResult(100, SimData, model2)
summary(Output2)

# Third analysis model: Model accounting for covariate with orthogonalization
ortho <- simFunction(residualCovariate, targetVar=1:6, covVar=7)
model3 <- model1
Output3 <- simResult(100, SimData, model3, objFunction=ortho)
param <- getPopulation(Output3)
param <- extract(param, y=1:6, e=1:2)
Output3 <- setPopulation(Output3, param)
summary(Output3)

# Fourth analysis model: Model accounting for covariate in factor level
loading <- matrix(0, 7, 3)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loading[7, 3] <- NA
path <- matrix(0, 3, 3)
path[2, 1] <- NA
path[1, 3] <- NA
path[2, 3] <- NA
errorCov <- diag(NA, 7)
errorCov[7, 7] <- 0
facCov <- diag(3)
Fac.Cov.Model <- simParamSEM(LY=loading, BE=path, TE=errorCov, PS=facCov)
model4 <- simModel(Fac.Cov.Model)
Output4 <- simResult(100, SimData, model4)

loadingVal <- matrix(0, 7, 3)
loadingVal[1:3, 1] <- 0.6
loadingVal[4:6, 2] <- 0.6
loadingVal[7, 3] <- 1
LY <- simMatrix(loading, loadingVal)
pathVal <- matrix(0, 3, 3)
pathVal[2, 1] <- 0.4
pathVal[1, 3] <- 0.4
pathVal[2, 3] <- 0.4
BE <- simMatrix(path, pathVal)
PS <- symMatrix(facCov)
errorCovVal <- diag(0.64, 7)
errorCovVal[7, 7] <- 0
TE <- symMatrix(errorCov, errorCovVal)
Fac.Cov.Model.Full <- simSetSEM(LY=LY, PS=PS, BE=BE, TE=TE)
Output4 <- setPopulation(Output4, Fac.Cov.Model.Full)
summary(Output4)
```

Remark

1) If we run an analysis based on real data, we will not find the column whether the confidence intervals of parameter estimates bracket population parameters. We can put the data generation population into the result of an analysis of a single data by the `setPopulation` function as well:

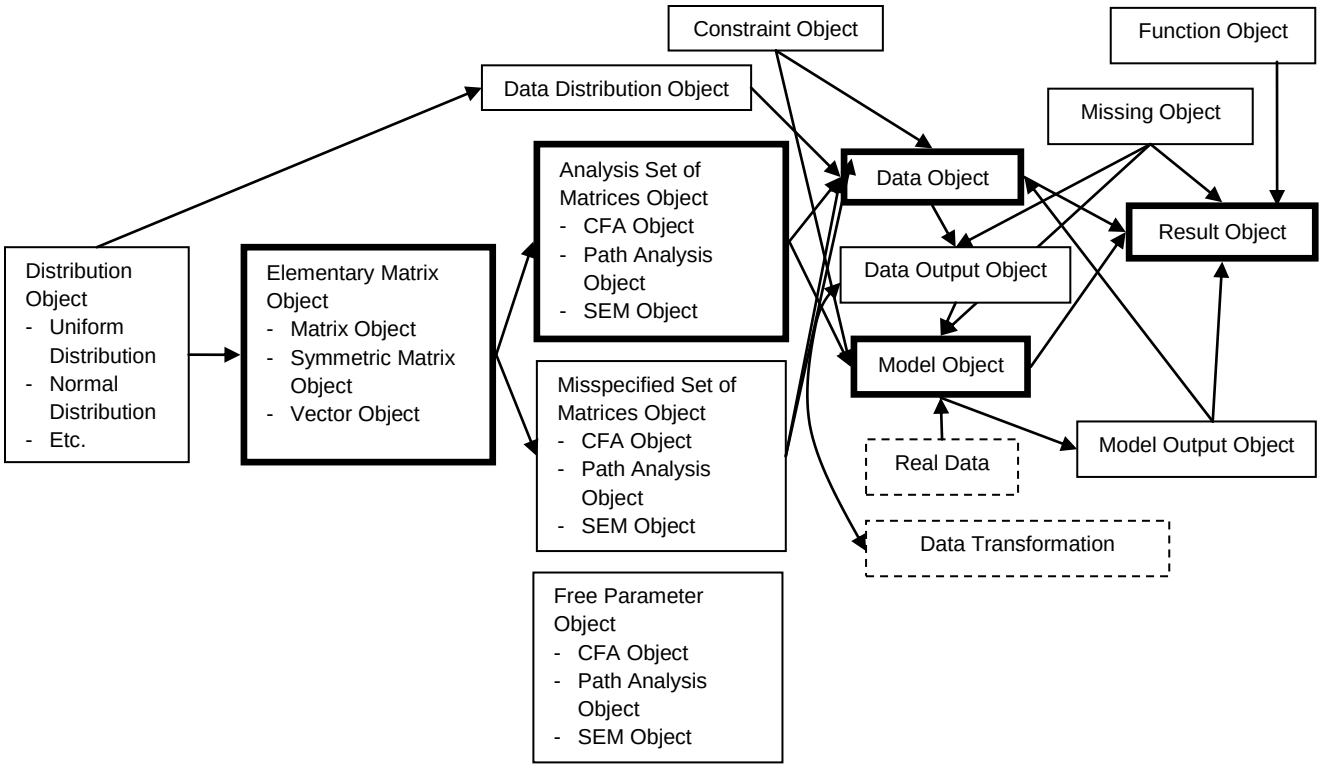
```
dat <- run(SimData)
out <- run(model4, dat)
out <- setPopulation(out, Fac.Cov.Model.Full)
```

Functions Review

Functions	Usage
`residualCovariate`	Orthogonalize the target variables based on covariates
`simFunction`	Save a specification of a function to be used after simulated data are generated.
`summaryPopulation`	Summarize the population model underlying the data generation process
`getPopulation`	Get the population model underlying the data generation process of each replication
`setPopulation`	Set a new population model for a simulation study, which can be used to compute biases in parameter estimates and standard errors.

Summary of Model Specification

This picture shows the map of all objects and their relationships in the package. All solid border boxes indicate the objects in the `simsem` package. The bold-border boxes shows all objects used in the Example 1, which are required objects for simulation. This is the minimal requirement to run a Monte Carlo simulation if you do not have real data. The dashed boxes indicate things that are not the object in this package but can interact with the package.



Accessing Help Files

Function

You can access help file in each function by

```
?run
```

Class

You can access help file in each class by

```
class?SimMatrix
```

Methods in each class

You can access help file for a method that uses in multiple classes by

```
method?run
```

If you would like to see help file of a method that uses in a specific class by

```
method?run("SimMatrix")
```

List of Distribution Objects

Here is the list of distribution objects that can be used in this package. The details of each attribute can be searched from the help of each constructor or the R help page of each distribution. This package uses the same names of attributes as in the R program.

Distribution	Class	Constructor	Attributes
Beta	SimBeta	simBeta	shape1, shape2, ncp
Binomial	SimBinom	simBinom	size, prob
Cauchy	SimCauchy	simCauchy	location, scale
Chi-squared	SimChisq	simChisq	df, ncp
Exponential	SimExp	simExp	rate
F	SimF	simF	df1, df2, ncp
Gamma	SimGamma	simGamma	shape, rate
Geometric	SimGeom	simGeom	prob
Hypergeometric	SimHyper	simHyper	m, n, k
Log Normal	SimLnorm	simLnorm	meanlog, sdlog
Logistic	SimLogis	simLogis	location, scale
Negative Binomial	SimNbinom	simNbinom	size, prob
Normal	SimNorm	simNorm	mean, sd
Poisson	SimPois	simPois	lambda
t	SimT	simT	df, ncp
Uniform	SimUnif	simUnif	min, max
Weibull	SimWeibull	simWeibull	shape, scale

There are six methods for the distribution objects:

1. summary: provide a description of an object
2. summaryShort: provide a brief distribution of an object
3. run: generate a random number from an object
4. plotDist: plot a distribution of an object
5. skew: Find a skewness of an object
6. kurtosis: Find an excessive kurtosis of an object.

Public Objects

This section will list all classes and functions relating to all objects in this package except the distribution objects.

Classes

Public Classes	Getting Documentation by
SimData	class?SimData
SimDataDist	class?SimDataDist
SimDataOut	class?SimDataOut
SimEqualCon	class?SimEqualCon
SimFunction	class?SimFunction
SimMatrix	class?SimMatrix
SimMissing	class?SimMissing
SimMisspec	class?SimMisspec
SimModel	class?SimModel

SimModelMIOut	class?SimModelMIOut
SimModelOut	class?SimModelOut
SimParam	class?SimParam
SimResult	class?SimResult
SimSet	class?SimSet
SimVector	class?SimVector
SymMatrix	class?SymMatrix

S4 Functions

Public Functions	Getting Documentation by	Available Classes
adjust	method?adjust	SimMatrix, SymMatrix, SimVector
anova	method?anova	SimModelOut, SimResult
createImpliedMACS	method?createImpliedMACS	SimModelOut, SimModelMIOut, SimDataOut
extract	method?extract	vector, matrix, SimMatrix, SimVector, SimSet, SimDataDist, SimParam
getCutoff	method?getCutoff	SimResult
getPower	method?getPower	SimResult
kurtosis	method?kurtosis	vector, VirtualDist
plotCutoff	method?plotCutoff	SimResult
plotDist	method?plotDist	SimDataDist
plotPower	method?plotPower	SimResult
pValue	method?pValue	(numeric, vector), (numeric, data.frame), (SimModelOut, SimResult)
run	method?run	SimData, SimMatrix, SimSet, SimMisspec, SimModel, SimVector, SymMatrix, SimMissing, SimDataDist
runFit	method?runFit	SimModel, SimModelOut
simData	method?simData	SimSet, SimModelOut
simModel	method?simModel	SimSet
skew	method?skew	vector, VirtualDist
summary	method?summary	All classes
summaryParam	method?summaryParam	SimResult
summaryShort	method?summaryShort	All classes

S3 Functions

Public Functions	Getting Documentation by
imposeMissing	?imposeMissing
indProd	?indProd
loadingFromAlpha	?loadingFromAlpha
miPoolChi	?miPoolChi
miPoolVector	?miPoolVector
residualCovariate	?residualCovariate
runMI	?runMI
simData	?simData
simEqualCon	?simEqualCon
simFunction	?simFunction
simMatrix	?simMatrix
simMissing	?simMissing
simMisspecCFA	?simMisspecCFA
simMisspecPath	?simMisspecPath
simMisspecSEM	?simMisspecSEM
simNorm	?simNorm
simParamCFA	?simParamCFA
simParamPath	?simParamPath
simParamSEM	?simParamSEM
simResult	?simResult
simSetCFA	?simSetCFA
simSetPath	?simSetPath
simSetSEM	?simSetSEM
simUnif	?simUnif

simVector	?simVector
symMatrix	?symMatrix

Symbols of Matrices

Here are the symbols of the matrices used to make the set of matrices object (simSetCFA, simSetPath, or simSetSEM) or the misspecified set of matrices object (simMisspecCFA, simMisspecPath, or simMisspecSEM). The matrix symbols can be classified into different categories:

1. Endogenous
    a. Covariance
        i. Among Factors: PS
        ii. Among Measurement Errors: TE
    b. Regression among Factors: BE
    c. Loading from Factors to Indicators: LY
    d. Correlation
        i. Among Factors: RPS
        ii. Among Measurement Errors: RTE
    e. Residual Variance
        i. Factors: VPS
        ii. Measurement Errors: VTE
    f. Total Variance
        i. Factors: VE
        ii. Measurement Errors: VY
    g. Intercept
        i. Factors: AL
        ii. Measurement Errors: TY
    h. Total Mean
        i. Factors: ME
        ii. Measurement Errors: MY
2. Exogenous
    a. Covariance
        i. Among Factors: PH
        ii. Among Measurement Errors: TD
    b. Loading from Factors to Indicators: LX
    c. Correlation
        i. Among Factors: RPH
        ii. Among Measurement Errors: RTD
    d. Residual Variance of Measurement Errors: VTD
    e. Total Variance
        i. Factors: VPH or VK
        ii. Measurement Errors: VX

   f. Intercept of Measurement Errors: TX
   g. Total Mean
      i. Factors: KA or MK
      ii. Measurement Errors: MX
3. Exogenous and Endogenous
   a. Covariance between Measurement Intercepts: TH
   b. Regression among Factors: GA
   c. Correlation between Measurement Intercepts: RTH

Fit Indices Details

Here is the list of fit indices provided in the simsem package:

1. Chi-square Test of the Target Model (χ^2_T). The value of chi-square is the -2 times log likelihood between the observed means and covariance matrix and model-implied means and covariance matrix. The degree of freedom (df_T) is the number of elements in the means and covariance matrix subtracted by the number of free parameters in the target model.
2. Chi-square Test of the Baseline Model (χ^2_B). Mostly, the baseline model estimates means and variances of the observed data but not the covariances of the observed data. When there are auxiliary variables, the covariance of the auxiliary variables to all other variables (including themselves) are estimated. The chi-square value is the -2 times log likelihood between the observed means and covariance matrix and baseline model-implied means and covariance matrix. The degree of freedom (df_B) is the number of elements in means and covariance matrix and the number of free parameters in the baseline model.
3. Comparative Fit Index (CFI). This index is one of the relative fit indices comparing between the fit of the target model and the fit of the baseline model. The minimum is 0 indicating bad fit and the maximum is 1 indicating perfect fit.

$$CFI = \frac{(\chi^2_B - df_B) - (\chi^2_T - df_T)}{\chi^2_B - df_B}$$

4. Tucker-Lewis Index (TLI) or Non-Normed Fit Index (NNFI). This index is also one of the relative fit indices comparing between target model and baseline model. The minimum is 0 indicating bad fit and the maximum can be slightly greater than 1. The larger value indicates good fit.

$$TLI = \frac{\frac{\chi^2_B}{df_B} - \frac{\chi^2_T}{df_T}}{\frac{\chi^2_B}{df_B} - 1}$$

5. Akaike Information Criterion (AIC). This index is usually used to compare between two nonnested model. The model with smaller AIC provides better fit to the observed data.

$$AIC = f_T + 2k_T$$

where $f_T = \exp(-\chi_T^2/2)$ and k_T is the number of free parameters

6. Bayesian Information Criterion (BIC). This index is also usually used to compare between two nonnested model. The model with smaller BIC provides better fit to the observed data.

$$BIC = f_T + \log(N)\, k_T$$

where *N* is sample size.

7. Root Mean Squared Error of Approximation (RMSEA). This index approximates the amount of misfit per degree of freedom. The minimum value is 0 indicating excellent fit.

$$RMSEA = \sqrt{\frac{\chi_T^2 - df_T}{df_T(N-1)}}$$

8. Standardized Root Mean Squared Residual (SRMR). This index indicates the average discrepancy between observed correlations and model-implied correlations. The minimum value is 0 indicating excellent fit.

$$SRMR = \sqrt{\frac{2\sum_i \sum_{j \le i} \left(\frac{s_{ij}}{\sqrt{s_{ii}}\sqrt{s_{jj}}} - \frac{\hat{\sigma}_{ij}}{\sqrt{\hat{\sigma}_{ii}}\sqrt{\hat{\sigma}_{jj}}} \right)^2}{p(p+1)}}$$

where s_{ij} is the observed covariance between indicator *i* and *j*, $\hat{\sigma}_{ij}$ is the model-implied covariance between the indicator *i* and *j*, and *p* is the number of indicators.

Give Us Feedback

If you found any bugs or had any suggestions, please let us know at

Sunthud Pornprasertmanit
Center for Research Methods and Data Analysis
University of Kansas
Email: psunthud@ku.edu