Package ‘simsem’

February 12, 2020

Type Package

Title SIMulated Structural Equation Modeling

Version 0.5-15

Date 2020-02-12

Maintainer Terrence D. Jorgensen <TJorgensen314@gmail.com>

Depends R(>= 3.3), methods, lavaan(>= 0.6.1), utils, stats, graphics

Suggests parallel, Amelia, quantreg, splines, foreign, KernSmooth, semTools(>= 0.4-15), OpenMx(>= 2.0.0), copula

Description Provides an easy framework for Monte Carlo simulation in structural equation modeling, which can be used for various purposes, such as such as model fit evaluation, power analysis, or missing data handling and planning.

License GPL (>= 2)

LazyLoad yes

URL http://www.simsem.org

NeedsCompilation no

Author Sunthud Pornprasertmanit [aut],
Patrick Miller [aut],
Alexander Schoemann [aut],
Terrence D. Jorgensen [aut, cre]
(<https://orcid.org/0000-0001-5111-6773>),
Corbin Quick [ctb]

Repository CRAN

Date/Publication 2020-02-12 16:40:02 UTC

R topics documented:

`analyze` *Data analysis using the model specification*

Description

Data analysis using the model specification (`linkS4class{SimSem}`) or the mx model object (`MxModel`). Data will be multiply imputed if the `miss` argument is specified.

Usage

```
analyze(model, data, package="lavaan", miss=NULL, aux=NULL, group = NULL,
mxMixture = FALSE, ...)
```

Arguments

`model`	The simsem model template (`linkS4class{SimSem}`) or the mx model object (`MxModel`)
`data`	The target dataset
`package`	The package used in data analysis. Currently, only `lavaan` package can be used.

miss	The missing object with the specification of auxiliary variable or the specification for the multiple imputation.
aux	List of auxiliary variables
group	A group variable. This argument is applicable only when the `model` argument is a `MxModel` object.
mxMixture	A logical whether to the analysis model is a mixture model. This argument is applicable when `MxModel` is used in the `model` argument only.
...	Additional arguments in the `lavaan` function. See also `lavOptions`

Value

The `lavaan` object containing the output

Author(s)

Patrick Miller (University of Notre Dame; <pmille13@nd.edu>), Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

Note that users can use functions provided by `lavaan` package (`lavaan`, `cfa`, `sem`, or `growth`) if they wish to analyze data by lavaan directly.

Examples

```
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
LY <- bind(loading, 0.7)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))

VY <- bind(rep(NA,6),2)

CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType = "CFA")

dat <- generate(CFA.Model,200)
out <- analyze(CFA.Model,dat)
```

anova *Provide a comparison of nested models and nonnested models across replications*

Description

This function will provide averages of model fit statistics and indices for nested models. It will also provide average differences of fit indices and power for likelihood ratio tests of nested models.

Arguments

`object`	`SimResult` object being described. Currently at least two objects must be included as arguments
`...`	any additional arguments, such as additional objects or for the function with result object

Value

A data frame that provides the statistics described above from all parameters. For using with `linkS4class{SimResult}`, the result is a list with two or three elements:

- `summary`: Average of fit indices across all replications
- `diff`: Average of the differences in fit indices across all replications
- `varyParam`: The statistical power of chi-square difference test given values of varying parameters (such as sample size or percent missing)

Author(s)

Alexander M. Schoemann (East Carolina University; <schoemanna@ecu.edu>), Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

`SimResult` for the object input

Examples

```
## Not run:
loading1 <- matrix(0, 6, 1)
loading1[1:6, 1] <- NA
loading2 <- loading1
loading2[6,1] <- 0
LY1 <- bind(loading1, 0.7)
LY2 <- bind(loading2, 0.7)
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model1 <- model(LY = LY1, RPS = RPS, RTE = RTE, modelType="CFA")
CFA.Model2 <- model(LY = LY2, RPS = RPS, RTE = RTE, modelType="CFA")
```

```
# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
# Need to make sure that both simResult calls have the same seed!
Output1 <- sim(5, n=500, model=CFA.Model1, generate=CFA.Model1, seed=123567)
Output2 <- sim(5, n=500, model=CFA.Model2, generate=CFA.Model1, seed=123567)
anova(Output1, Output2)

# The example when the sample size is varying
Output1b <- sim(NULL, n=seq(50, 500, 50), model=CFA.Model1, generate=CFA.Model1, seed=123567)
Output2b <- sim(NULL, n=seq(50, 500, 50), model=CFA.Model2, generate=CFA.Model1, seed=123567)
anova(Output1b, Output2b)

## End(Not run)
```

---

bind *Specify matrices for Monte Carlo simulation of structural equation models*

Description

Create SimMatrix or SimVector object that specifies

1. Pattern of fixed/freed parameters for analysis
2. Population parameter values for data generation
3. Any model misspecification (true population parameter is different than the one specified) for these parameters.

Each matrix in the Lisrel-style notation is specified in this way (e.g. LY, PS, and TE) and is used to create a model analysis template and a data generation template for simulation through the `model` function.

Usage

```
bind(free = NULL, popParam = NULL, misspec = NULL, symmetric = FALSE)
binds(free = NULL, popParam = NULL, misspec = NULL, symmetric = TRUE)
```

Arguments

`free` Required matrix or vector where each element represents a fixed or freed parameter used for analysis with structural equation models. Parameters can be freed by setting the corresponding element in the matrix to `NA`, and can be fixed by setting the value of the element to any number (e.g. 0). Parameters can be labeled using any character string. Any labeled parameter is considered to be free, and parameters with identical labels will be constrained to equality for analysis.

`popParam` Optional matrix or vector of identical dimension to the free matrix whose elements contain population parameter values for data generation in simulation. For simlutation, each free parameter requires a population parameter value, which is a quoted numeric value. Parameters that don't have population values are left as empty strings. Population parameters can also be drawn from a distribution. This is done by wrapping a call to create 1 value from an existing random generation function in quotes: e.g `"runif(1,0,1)","rnorm(1,0,.01)"` Every replication in the simulation will draw a parameter value from this distribution. The function checks that what is quoted is valid R.

If a random population parameter is constrained to equality in the free matrix, *each drawn population parameter value will be the same.* More details on data generation is available in `?generate`, `?createData`, and `?draw`.

To simplify the most common case, `popParam` can take 1 value or distribution and create a matrix or vector that assigns that population parameter or distribution to all freed parameters. These population values are used as starting values for analysis by default.

`misspec` Optional matrix or vector of identical dimension to the free matrix whose elements contain population parameter values for specifying misspecification. Elements of the misspec matrix contain population parameters that are added to parameters that are fixed or have an existing population value. These parameters are also quoted numeric strings, and can optionally be drawn from distributions as described above. To simplify the most common case, misspec can take 1 value or distribution and create a matrix or vector that assigns that value or distribution to all previously specified fixed parameters. Details about misspecification are included the data generation functions.

`symmetric` Set as `TRUE` if the matrix created is symmetric (RPS/PS, RTE/TE). The function `binds` can also be used, which defaults to `symmetric = TRUE`

Details

Bind is the first step in the `bind` -> `model` -> `sim` workflow of *simsem*, and this document outlines the user interface or language used to describe these simulations. This interface, while complex, enables a wide array of simulation specifications for structural equation models by building on LISREL-style parameter specifications.

In simulations supported by *simsem*, a given parameter may be either fixed or freed for analysis, but may optionally also have a population value or distribution for data generation, or a value or distribution of misspecification. The purpose of bind is to stack these multiple meanings of a parameter into an object recognized by *simsem*, a `SimMatrix`. Each matrix in the Lisrel notation (e.g. LY, PS, TE, BE) becomes a `SimMatrix`, and is passed to the function `model`, which builds the data generation template and an analysis template (a lavaan parameter table), collectively forming a `SimSem` object, which can be passed to the function `sim` for simulation.

Value

`SimMatrix` or `SimVector` object that used for model specification for analysis and data generation in simsem.

Author(s)

Patrick Miller (University of Notre Dame; <pmille13@nd.edu>), Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `model` To combine simMatrix objects into a complete data analysis and data generation template, which is a `SimSem` object
- `generate` To generate data using the simsem template.
- `analyze` To analyze real or generated data using the simsem template.

Examples

```
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loadingValues <- matrix(0, 6, 2)
loadingValues[1:3, 1] <- 0.7
loadingValues[4:6, 2] <- 0.7
LY <- bind(loading, loadingValues)
summary(LY)

# Set both factor correlations to .05
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

# Misspecify all error covarainces
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- NA
RTE <- binds(error.cor,1,"runif(1,-.05,.05)")
```

---

bindDist *Create a data distribution object.*

Description

Create a data distribution object. There are two ways to specify nonnormal data-generation model. To create nonnormal data by the copula method, `margins` and `...` arguments are required. To create data by Vale and Maurelli's method, `skewness` and/or `kurtosis` arguments are required.

Usage

```
bindDist(margins = NULL, ..., p = NULL, keepScale = TRUE, reverse = FALSE,
copula = NULL, skewness = NULL, kurtosis = NULL)
```

Arguments

- `margins`: A character vector specifying all the marginal distributions. The characters in argument margins are used to construct density, distribution, and quantile function names. For example, `"norm"` can be used to specify marginal distribution, because `"dnorm"`, `"pnorm"`, and `"qnorm"` are all available. A user-defined distribution or other distributions can be used. For example, `"gl"` function in the `"gld"` package can be used to represent the generalized lambda distribution where `"dgl"`, `"pgl"`, and `"qgl"` are available. See the description of `margins` attribute of the `Mvdc` function for further details.
- `...`: A list whose each component is a list of named components, giving the parameter values of the marginal distributions. See the description of `paramMargins` attribute of the `Mvdc` function for further details.
- `p`: Number of variables. If only one distribution object is listed, the `p` will make the same distribution objects for all variables.
- `keepScale`: A vector representing whether each variable is transformed its mean and standard deviation or not. If TRUE, transform back to retain the mean and standard deviation of a variable equal to the model implied mean and standard deviation (with sampling error)
- `reverse`: A vector representing whether each variable is mirrored or not. If `TRUE`, reverse the distribution of a variable (e.g., from positive skewed to negative skewed. If one logical value is specified, it will apply to all variables.
- `copula`: A copula class that represents the multivariate distribution, such as `ellipCopula`, `normalCopula`, or `archmCopula`. When this copula argument is specified, the data-transformation method from Mair, Satorra, and Bentler (2012) is used. If this copula argument is not specified, the naive Gaussian copula is used such that the correlation matrix is direct applied to the multivariate Gaussian copula. The correlation matrix will be equivalent to the Spearman's correlation (rank correlation) of the resulting data.
- `skewness`: A vector of skewness of each variable. The Vale & Maurelli (1983) method is used in data generation.
- `kurtosis`: A vector of (excessive) kurtosis of each variable. The Vale & Maurelli (1983) method is used in data generation.

Value

`SimDataDist` that saves analysis result from simulate data.

Author(s)

Sunthud Pornprasertmanit (<`psunthud@gmail.com`>)

References

Mair, P., Satorra, A., & Bentler, P. M. (2012). Generating nonnormal multivariate data using copulas: Applications to SEM. *Multivariate Behavioral Research, 47*, 547-565.

Vale, C. D. & Maurelli, V. A. (1983) Simulating multivariate nonormal distributions. *Psychometrika, 48*, 465-471.

See Also

- SimResult for the type of resulting object

Examples

```
# Create data based on Vale and Maurelli's method by specifying skewness and kurtosis
dist <- bindDist(skewness = c(0, -2, 2), kurtosis = c(0, 8, 4))

## Not run:
library(copula)

# Create three-dimensional distribution by gaussian copula with
# the following marginal distributions
#   1. t-distribution with df = 2
#  2. chi-square distribution with df = 3
# 3. normal distribution with mean = 0 and sd = 1

# Setting the attribute of each marginal distribution
d1 <- list(df=2)
d2 <- list(df=3)
d3 <- list(mean=0, sd=1)

# Create a data distribution object by setting the names of each distribution
# and their arguments
dist <- bindDist(c("t", "chisq", "norm"), d1, d2, d3)

# Create data by using Gumbel Copula as the multivariate distribution
dist <- bindDist(c("t", "chisq", "norm"), d1, d2, d3, copula = gumbelCopula(2, dim = 3))

# Reverse the direction of chi-square distribution from positively skew to negatively skew
dist <- bindDist(c("t", "chisq", "norm"), d1, d2, d3, copula = gumbelCopula(2, dim = 3),
reverse = c(FALSE, TRUE, FALSE))

## End(Not run)
```

---

coef — *Extract parameter estimates from a simulation result*

Description

Extract parameter estimates from a simulation result. This function is similar to the inspect method with what = "coef".

Arguments

object	The target SimResult object
improper	Specify whether to include the information from the replications with improper solutions
nonconverged	Specify whether to include the information from the nonconvergent replications

Value

Parameter estimates of each replication

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

SimResult for the object input

Examples

```
## Not run:
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
LY <- bind(loading, 0.7)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))

VY <- bind(rep(NA,6),2)

CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType = "CFA")

# In reality, more than 5 replications are needed.
Output <- sim(5, CFA.Model, n=200)
coef(Output)
coef(Output, improper = TRUE)

## End(Not run)
```

---

combineSim *Combine result objects*

Description

Combine result objects into a single result object

Usage

```
combineSim(...)
```

Arguments

...	Result objects, SimResult

Value

A combined result object

Author(s)

Terry Jorgensen (University of Kansas; <TJorgensen314@gmail.com>), Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

Result object (SimResult)

Examples

```
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
LY <- bind(loading, 0.7)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))

VY <- bind(rep(NA,6),2)

CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType = "CFA")
Output1 <- sim(5, CFA.Model, n=200, seed=123321)
Output2 <- sim(4, CFA.Model, n=200, seed=324567)
Output3 <- sim(3, CFA.Model, n=200, seed=789987)
Output <- combineSim(Output1, Output2, Output3)
summary(Output)
```

---

continuousCoverage *Find coverage rate of model parameters when simulations have randomly varying parameters*

Description

A function to find the coverage rate of confidence intervals in a model when one or more of the simulations parameters vary randomly across replications.

Usage

```
continuousCoverage(simResult, coverValue = NULL, contN = TRUE, contMCAR = FALSE,
    contMAR = FALSE, contParam = NULL, coverParam = NULL, pred = NULL)
```

Arguments

simResult	SimResult that includes at least one randomly varying parameter (e.g. sample size, percent missing, model parameters)
coverValue	A target value used that users wish to find the coverage rate of that value (e.g., 0). If NULL, the parameter values will be used.
contN	Logical indicating if N varies over replications.
contMCAR	Logical indicating if the percentage of missing data that is MCAR varies over replications.
contMAR	Logical indicating if the percentage of missing data that is MAR varies over replications.
contParam	Vector of parameters names that vary over replications.
coverParam	Vector of parameters names that the user wishes to find coverage rate for. This can be a vector of names (e.g., "f1=~y2", "f1~~f2"). If parameters are not specified, coverage rates for all parameters in the model will be returned.
pred	A list of varying parameter values that users wish to find statistical power from.

Details

In this function, the coverage (which can be 0 or 1) is regressed on randomly varying simulation parameters (e.g., sample size, percentage of missing data, or model parameters) using logistic regression. For a set of independent variables values, the predicted probability from the logistic regression equation is the predicted coverage rate.

Value

Data frame containing columns representing values of the randomly varying simulation parameters, and coverage rates for model parameters of interest.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>), Alexander M. Schoemann (East Carolina University; <schoemanna@ecu.edu>)

See Also

- SimResult to see how to create a simResult object with randomly varying parameters.

Examples

```
## Not run:
# Specify Sample Size by n
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, 0.7)
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")
```

```
# Specify both continuous sample size and percent missing completely at random.
# Note that more fine-grained values of n and pmMCAR is needed, e.g., n=seq(50, 500, 1)
# and pmMCAR=seq(0, 0.2, 0.01)
Output <- sim(NULL, CFA.Model, n=seq(100, 200, 20), pmMCAR=c(0, 0.1, 0.2))
summary(Output)

# Find the coverage rates of all combinations of different sample size and percent MCAR missing
Ccover <- continuousCoverage(Output, contN = TRUE, contMCAR = TRUE)
Ccover

# Find the coverage rates of parameter estimates when sample size is 200
# and percent MCAR missing is 0.3
Ccover2 <- continuousCoverage(Output, coverValue=0, contN = TRUE, contMCAR = TRUE,
     pred=list(N = 200, pmMCAR = 0.3))
Ccover2

## End(Not run)
```

`continuousPower` *Find power of model parameters when simulations have randomly varying parameters*

Description

A function to find the power of parameters in a model when one or more of the simulations parameters vary randomly across replications.

Usage

```
continuousPower(simResult, contN = TRUE, contMCAR = FALSE, contMAR = FALSE,
contParam = NULL, alpha = .05, powerParam = NULL, pred = NULL)
```

Arguments

`simResult`	`SimResult` that includes at least one randomly varying parameter (e.g. sample size, percent missing, model parameters)
`contN`	Logical indicating if N varies over replications.
`contMCAR`	Logical indicating if the percentage of missing data that is MCAR varies over replications.
`contMAR`	Logical indicating if the percentage of missing data that is MAR varies over replications.
`contParam`	Vector of parameters names that vary over replications.
`alpha`	Alpha level to use for power analysis.
`powerParam`	Vector of parameters names that the user wishes to find power for. This can be a vector of names (e.g., "f1=~y2", "f1~~f2"). If parameters are not specified, power for all parameters in the model will be returned.
`pred`	A list of varying parameter values that users wish to find statistical power from.

Details

A common use of simulations is to conduct power analyses, especially when using SEM (Muthen & Muthen, 2002). Here, researchers select values for each parameter and a sample size and run a simulation to determine power in those conditions (the proportion of generated datasets in which a particular parameter of interest is significantly different from zero). To evaluate power at multiple sample sizes, one simulation for each sample size must be run. By continuously varying sample size across replications, only a single simulation is needed. In this simulation, the sample size for each replication varies randomly across plausible sample sizes (e.g., sample sizes between 200 and 500). For each replication, the sample size and significance of each parameter (0 = not significant, 1 = significant) are recorded. When the simulation is complete, parameter significance is regressed on sample size using logistic regression. For a given sample size, the predicted probability from the logistic regression equation is the power to detect an effect at that sample size. This approach can be extended to other randomly varying simulation parameters such as the percentage of missing data, and model parameters.

Value

Data frame containing columns representing values of the randomly varying simulation parameters, and power for model parameters of interest.

Author(s)

Alexander M. Schoemann (East Carolina University; <schoemanna@ecu.edu>), Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

Muthen, L. K., & Muthen, B. O. (2002). How to use a Monte Carlo study to decide on sample size and determine power. *Structural Equation Modeling, 4,* 599-620.

See Also

- `SimResult` to see how to create a simResult object with randomly varying parameters.

Examples

```
## Not run:
# Specify Sample Size by n
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, 0.7)
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")
dat <- generate(CFA.Model, 50)
out <- analyze(CFA.Model, dat)

# Specify both continuous sample size and percent missing completely at random.
# Note that more fine-grained values of n and pmMCAR is needed, e.g., n=seq(50, 500, 1)
# and pmMCAR=seq(0, 0.2, 0.01)
```

```
Output <- sim(NULL, CFA.Model, n=seq(100, 200, 20), pmMCAR=c(0, 0.1, 0.2))
summary(Output)

# Find the power of all combinations of different sample size and percent MCAR missing
Cpow <- continuousPower(Output, contN = TRUE, contMCAR = TRUE)
Cpow

# Find the power of parameter estimates when sample size is 200 and percent MCAR missing is 0.3
Cpow2 <- continuousPower(Output, contN = TRUE, contMCAR = TRUE, pred=list(N = 200, pmMCAR = 0.3))
Cpow2

## End(Not run)
```

`createData` *Create data from a set of drawn parameters.*

Description

This function can be used to create data from a set of parameters created from `draw`, called a code-paramSet. This function is used internally to create data, and is available publicly for accessibility and debugging.

Usage

```
createData(paramSet, n, indDist=NULL, sequential=FALSE, facDist=NULL,
errorDist=NULL, saveLatentVar = FALSE, indLab=NULL, facLab = NULL,
modelBoot=FALSE, realData=NULL, covData=NULL, empirical = FALSE)
```

Arguments

`paramSet`	Set of drawn parameters from `draw`.
`n`	Integer of desired sample size.
`indDist`	A `SimDataDist` object or list of objects for a distribution of indicators. If one object is passed, each indicator will have the same distribution. Use when `sequential` is `FALSE`.
`sequential`	If `TRUE`, use a sequential method to create data such that the data from factor are generated first and apply to a set of equations to obtain the data of indicators. If `FALSE`, create data directly from model-implied mean and covariance of indicators.
`facDist`	A `SimDataDist` object or list of objects for the distribution of factors. If one object is passed, all factors will have the same distribution. Use when `sequential` is `TRUE`.
`errorDist`	An object or list of objects of type `SimDataDist` indicating the distribution of errors. If a single `SimDataDist` is specified, each error will be genrated with that distribution.

saveLatentVar
: If TRUE, the total latent variable scores, residual latent variable scores, and measurement error scores are also provided as the "latentVar" attribute of the generated data by the following line: attr(generatedData,"latentVar"). The sequential argument must be TRUE in order to use this option.

indLab
: A vector of indicator labels. When not specified, the variable names are x1,x2,... xN.

facLab
: A vector of factor labels. When not specified, the variable names are f1,f2,... fN.

modelBoot
: When specified, a model-based bootstrap is used for data generation. See details for further information. This argument requires real data to be passed to readData.

realData
: A data.frame containing real data. The data generated will follow the distribution of this data set.

covData
: A data.frame containing covariate data, which can have any distributions. This argument is required when users specify GA or KA matrices in the model template (SimSem).

empirical
: Logical. If TRUE, the specified parameters are treated as sample statistics and data are created to get the specified sample statistics. This argument is applicable when multivariate normal distribution is specified only.

Details

This function will use the modified mvrnorm function (from the MASS package) by Paul E. Johnson to create data from model implied covariance matrix if the data distribution object (SimDataDist) is not specified. The modified function is just a small modification from the original mvrnorm function such that the data generated with the sample sizes of n and n + k (where k > 0) will be replicable in the first n rows.

It the data distribution object is specified, either the copula model or the Vale and Maurelli's method is used. For the copula approach, if the copula argument is not specified in the data distribution object, the naive Gaussian copula is used. The correlation matrix is direct applied to the multivariate Gaussian copula. The correlation matrix will be equivalent to the Spearman's correlation (rank correlation) of the resulting data. If the copula argument is specified, such as ellipCopula, normalCopula, or archmCopula, the data-transformation method from Mair, Satorra, and Bentler (2012) is used. In brief, the data (X) are created from the multivariate copula. The covariance from the generated data is used as the starting point (S). Then, the target data (Y) with the target covariance as model-implied covariance matrix (Σ_0) can be created:

$$Y = XS^{-1/2}\Sigma_0^{1/2}.$$

See bindDist for further details. For the Vale and Maurelli's (1983) method, the code is brought from the lavaan package.

For the model-based bootstrap, the transformation proposed by Yung & Bentler (1996) is used. This procedure is the expansion from the Bollen and Stine (1992) bootstrap including a mean structure. The model-implied mean vector and covariance matrix with trivial misspecification will be used in the model-based bootstrap if misspec is specified. See page 133 of Bollen and Stine (1992) for a reference.

Internally, parameters are first drawn, and data is then created from these parameters. Both of these steps are available via the `draw` and `createData` functions respectively.

Value

A data.frame containing simulated data from the data generation template. A variable "group" is appended indicating group membership.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>), Patrick Miller (University of Notre Dame; <pmille13@nd.edu>). The original code of `mvrnorm` function is based on the `MASS` package slightly modified by Paul E. Johnson. The code for data-transformation in multivariate copula is based on Mair et al. (2012) article. The code for Vale and Maurelli (1983) is slightly modified from the function provided in the `lavaan` package.

References

Bollen, K. A., & Stine, R. A. (1992). Bootstrapping goodness-of-fit measures in structural equation models. *Sociological Methods and Research, 21,* 205-229.

Mair, P., Satorra, A., & Bentler, P. M. (2012). Generating nonnormal multivariate data using copulas: Applications to SEM. *Multivariate Behavioral Research, 47*, 547-565.

Vale, C. D. & Maurelli, V. A. (1983) Simulating multivariate nonormal distributions. *Psychometrika, 48*, 465-471.

Yung, Y.-F., & Bentler, P. M. (1996). Bootstrapping techniques in analysis of mean and covariance structures. In G. A. Marcoulides & R. E. Schumacker (Eds.), *Advanced structural equation modeling: Issues and techniques* (pp. 195-226). Mahwah, NJ: Erlbaum.

Examples

```
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
LY <- bind(loading, 0.7)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))

VY <- bind(rep(NA,6),2)

CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType = "CFA")

# Draw a parameter set for data generation.
param <- draw(CFA.Model)

# Generate data from the first group in the paramList.
dat <- createData(param[[1]], n = 200)
```

draw *Draw parameters from a SimSem object.*

Description

This function draws parameters from a SimSem template, for debugging or other use. Used internally to create data. Data can be created in one step from a SimSem object using generate.

Usage

```
draw(model, maxDraw=50, misfitBounds=NULL, averageNumMisspec=FALSE,
optMisfit = NULL, optDraws = 50, misfitType = "f0", createOrder = c(1, 2, 3),
covData = NULL)
```

Arguments

model	A SimSem object.
maxDraw	Integer specifying the maximum number of attempts to draw a valid set of parameters (no negative error variance, standardized coefficients over 1).
misfitBounds	Vector that contains upper and lower bounds of the misfit measure. Sets of parameters drawn that are not within these bounds are rejected.
averageNumMisspec	If TRUE, the provided fit will be divided by the number of misspecified parameters.
optMisfit	Character vector of either "min" or "max" indicating either maximum or minimum optimized misfit. If not null, the set of parameters out of the number of draws in "optDraws" that has either the maximum or minimum misfit of the given misfit type will be returned.
optDraws	Number of parameter sets to draw if optMisfit is not null. The set of parameters with the maximum or minimum misfit will be returned.
misfitType	Character vector indicating the fit measure used to assess the misfit of a set of parameters. Can be "f0", "rmsea", "srmr", or "all".
createOrder	The order of 1) applying equality/inequality constraints, 2) applying misspecification, and 3) fill unspecified parameters (e.g., residual variances when total variances are specified). The specification of this argument is a vector of different orders of 1 (constraint), 2 (misspecification), and 3 (filling parameters). For example, c(1,2,3) is to apply constraints first, then add the misspecification, and finally fill all parameters.
covData	A data.frame containing covariate data, which can have any distributions. This argument is required when users specify GA or KA matrices in the model template (SimSem).

Value

Nested list of drawn parameters in the form `[[Group]][[param,misspec,misOnly]][[SimMatrix]]`. E.g. The LY parameter matrix of the first group would be indexed as `obj[[1]]$param$LY`. The values in $param are the raw parameter values with no misspecification. The values in $misspec are raw parameter values + misspecification. The values in $misOnly are only the misspecification values.

Author(s)

Sunthud Pornprasertmanit (<`psunthud@gmail.com`>), Patrick Miller (University of Notre Dame; <`pmille13@nd.edu`>)

See Also

`createData` To generate random data using a set of parameters from `draw`

Examples

```
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
LY <- bind(loading, 0.7)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))

VY <- bind(rep(NA,6),2)

CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType = "CFA")

# Draw a parameter set for data generation.
param <- draw(CFA.Model)

# Example of Multiple Group Model with Weak Invariance

loading.in <- matrix(0, 6, 2)
loading.in[1:3, 1] <- c("load1", "load2", "load3")
loading.in[4:6, 2] <- c("load4", "load5", "load6")
mis <- matrix(0,6,2)
mis[loading.in == "0"] <- "runif(1, -0.1, 0.1)"
LY.in <- bind(loading.in, "runif(1, 0.7, 0.8)", mis)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))
```

```
VTE <- bind(rep(NA, 6), 0.51)

VPS1 <- bind(rep(1, 2))

VPS2 <- bind(rep(NA, 2), c(1.1, 1.2))

# Inequality constraint
script <- "
sth := load1 + load2 + load3
load4 == (load5 + load6) / 2
load4 > 0
load5 > 0
sth2 := load1 - load2
"

# Model Template
weak <- model(LY = LY.in, RPS = RPS, VPS=list(VPS1, VPS2), RTE = RTE, VTE=VTE, ngroups=2,
     modelType = "CFA", con=script)

# Constraint --> Misspecification --> Fill Parameters
draw(weak, createOrder=c(1, 2, 3))

# Constraint --> Fill Parameters --> Misspecification
draw(weak, createOrder=c(1, 3, 2))

# Misspecification --> Constraint --> Fill Parameters
draw(weak, createOrder=c(2, 1, 3))

# Misspecification --> Fill Parameters --> Constraint
draw(weak, createOrder=c(2, 3, 1))

# Fill Parameters --> Constraint --> Misspecification
draw(weak, createOrder=c(3, 1, 2))

# Fill Parameters --> Misspecification --> Constraint
draw(weak, createOrder=c(3, 2, 1))
```

`estmodel` *Shortcut for data analysis template for simulation.*

Description

Creates a data analysis template (lavaan parameter table) for simulations with structural equation models based on Y-side LISREL design matrices. Each corresponds to a LISREL matrix, but must be a matrix or a vector. In addition to the usual Y-side matrices in LISREL, both PS and TE can be specified using correlations (RPS, RTE) and scaled by a vector of residual variances (VTE, VPS) or total variances (VY, VE). Multiple groups are supported by passing lists of matrices or vectors to arguments, or by specifying the number of groups.

Usage

```
estmodel(LY = NULL, PS = NULL, RPS = NULL, TE = NULL, RTE = NULL, BE = NULL,
VTE = NULL, VY = NULL, VPS = NULL, VE = NULL, TY = NULL, AL = NULL,
MY = NULL, ME = NULL, KA = NULL, GA = NULL, modelType, indLab = NULL,
facLab = NULL, covLab = NULL, groupLab = "group", ngroups = 1, con = NULL)
estmodel.cfa(LY = NULL,PS = NULL,RPS = NULL, TE = NULL,RTE = NULL, VTE = NULL,
VY = NULL, VPS = NULL, VE = NULL, TY = NULL, AL = NULL, MY = NULL, ME = NULL,
KA = NULL, GA = NULL, indLab = NULL, facLab = NULL, covLab = NULL,
groupLab = "group", ngroups = 1, con = NULL)
estmodel.path(PS = NULL, RPS = NULL, BE = NULL, VPS = NULL, VE = NULL, AL = NULL,
ME = NULL, KA = NULL, GA = NULL, indLab = NULL, facLab = NULL, covLab = NULL,
groupLab = "group", ngroups = 1,con = NULL)
estmodel.sem(LY = NULL,PS = NULL,RPS = NULL, TE = NULL,RTE = NULL, BE = NULL,
VTE = NULL, VY = NULL, VPS = NULL, VE = NULL, TY = NULL, AL = NULL, MY = NULL,
ME = NULL, KA = NULL, GA = NULL, indLab = NULL, facLab = NULL, covLab = NULL,
groupLab = "group", ngroups = 1, con = NULL)
```

Arguments

`LY`	Factor loading matrix from endogenous factors to Y indicators (need to be a matrix or a list of matrices).
`PS`	Residual covariance matrix among endogenous factors (need to be a symmetric matrix or a list of symmetric matrices).
`RPS`	Residual correlation matrix among endogenous factors (need to be a symmetric matrix or a list of symmetric matrices).
`TE`	Measurement error covariance matrix among Y indicators (need to be a symmetric matrix or a list of symmetric matrices).
`RTE`	Measurement error correlation matrix among Y indicators (need to be a symmetric matrix or a list of symmetric matrices).
`BE`	Regression coefficient matrix among endogenous factors (need to be a matrix or a list of matrices).
`VTE`	Measurement error variance of indicators (need to be a vector or a list of vectors).
`VY`	Total variance of indicators (need to be a vector or a list of vectors). NOTE: Either measurement error variance or indicator variance is specified. Both cannot be simultaneously specified.
`VPS`	Residual variance of factors (need to be a vector or a list of vectors).
`VE`	Total variance of of factors (need to be a vector or a list of vectors). NOTE: Either residual variance of factors or total variance of factors is specified. Both cannot be simulatneously specified.
`TY`	Measurement intercepts of Y indicators. (need to be a vector or a list of vectors).
`AL`	Endogenous factor intercept (need to be a vector or a list of vectors).
`MY`	Overall Y indicator means. (need to be a vector or a list of vectors). NOTE: Either measurement intercept of indicator mean can be specified. Both cannot be specified simultaneously.

ME — Total mean of endogenous factors (need to be a vector or a list of vectors). NOTE: Either endogenous factor intercept or total mean of endogenous factor is specified. Both cannot be simultaneously specified.

KA — Regression coefficient matrix from covariates to indicators (need to be a matrix or a list of matrices). KA is needed when (fixed) exogenous covariates are needed only.

GA — Regression coefficient matrix from covariates to factors (need to be a matrix or a list of matrices). GA is needed when (fixed) exogenous covariates are needed only.

modelType — "CFA", "Sem", or "Path". This is specified to ensure that the analysis and data generation template created based on specified matrices in model correspond to what the user intends.

indLab — Character vector of indicator labels. If left blank, automatic labels will be generated as `y1`, `y2`, ... `yy`.

facLab — Character vector of factor labels. If left blank, automatic labels will be generated as `f1`, `f2`, ... `ff`

covLab — Character vector of covariate labels. If left blank, automatic labels will be generated as `z1`, `z2`, ... `zz`

groupLab — Character of group-variable label (not the names of each group). If left blank, automatic labels will be generated as `group`

ngroups — Integer. Number of groups for data generation, defaults to 1. If larger than one, all specified matrices will be repeated for each additional group. If any matrix argument is a list, the length of this list will be the number of groups and ngroups is ignored.

con — Additional parameters (phantom variables), equality constraints, and inequality constraints that users wish to specify in the model. The additional parameters are specified in lavaan syntax. The allowed operator are ":=" (is defined as), "==" (is equal to), "<" (is less than), and ">" (is greater than). Names used in the syntax are the labels defined on free parameters in the model except that the left-handed-side name of ":=" is a new parameter name. On the right hand side of all operators, any mathematical expressions are allowed, e.g., `"newparam := (load1 + load2 + load3)/3"`. For the "<" and ">" operators in data generation, if the parameters relation is not hold (e.g., the left hand side is less than the right hand side in the ">" operator), the left hand side parameters will be changed such that the relation holds with a very small difference (i.e., 0.000001). For example, in "load1 > load2", if load1 is 0.5 and load2 is 0.6, load1 will be changed to 0.6 + 0.000001 = 0.600001.

Details

This function contains default settings:

For `modelType="CFA"`, `LY` is required. As the default, the on-diagonal elements of `PS` are fixed as 1 and the off-diagonal elements of `PS` are freely estimated. The off-diagonal elements of `TE` are freely estimated and the off-diagonal elements of `TE` are fixed to 0. The `AL` elements are fixed to 0. The `TY` elements are freely estimated.

For modelType="Path", BE is required. As the default, the on-diagonal elements of PS are freely estimated, the off-diagonal elements between exogenous variables (covariance between exogenous variables) are freely estimated, and the other off-diagonal elements are fixed to 0. The AL elements are freely estimated.

For modelType="SEM", LY and BE are required. As the default, the on-diagonal elements of PS are fixed to 1, the off-diagonal elements between exogenous factors (covariance between exogenous factors) are freely estimated, and the other off-diagonal elements are fixed to 0. The off-diagonal elements of TE are freely estimated and the off-diagonal elements of TE are fixed to 0. The AL elements are fixed to 0. The TY elements are freely estimated.

The estmodel.cfa, estmodel.path, and estmodel.sem are the shortcuts for the estmodel function when modelType are "CFA", "Path", and "SEM", respectively.

Value

SimSem object that contains the data generation template (@dgen) and analysis template (@pt).

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- model To build data generation and data analysis template for simulation.
- sim For simulations using the SimSem template.
- generate To generate data using the SimSem template.
- analyze To analyze real or generated data using the SimSem template.
- draw To draw parameters using the SimSem template.

Examples

```
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

CFA.Model <- estmodel(LY = loading, modelType = "CFA")

path <- matrix(0, 4, 4)
path[3, 1:2] <- NA
path[4, 3] <- NA
Path.Model <- estmodel(BE = path, modelType = "Path")

SEM.Model <- estmodel(BE = path, LY = loading, modelType="SEM")

# Shortcut
CFA.Model <- estmodel.cfa(LY = loading)
Path.Model <- estmodel.path(BE = path)
SEM.Model <- estmodel.sem(BE = path, LY = loading)
```

exportData *Export data sets for analysis with outside SEM program.*

Description

This function can be used to export data created from a set of parameters created from draw, called a codeparamSet. This function can export data to be analyzed with either Mplus or LISREL.

Usage

```
exportData(nRep, model, n, program = "Mplus", fileStem = "sim", miss = NULL,
missCode = -999, datafun=NULL, pmMCAR = NULL, pmMAR = NULL, facDist = NULL,
indDist = NULL, errorDist = NULL, sequential = FALSE, modelBoot = FALSE,
realData = NULL, maxDraw = 50, misfitType = "f0", misfitBounds = NULL,
averageNumMisspec = NULL, optMisfit=NULL, optDraws = 50, seed = 123321,
silent = FALSE, multicore = FALSE, numProc = NULL,  params = FALSE)
```

Arguments

nRep	Number of replications. Users can specify as NULL and specify n, pmMCAR, and pmMAR
model	SimSem object created by model. Will be used to generate data and analyze it.
n	Sample size. This argument is not necessary except the user wish to vary sample size across replications. The sample size here is a vector of sample size in integers. For the random distribution object, if the resulting value has decimal, the value will be rounded.
program	Statistical program that will be used to analyze data. Currently only Mplys and LISREL are supported.
fileStem	The stem of the filename(s) for file(s) output. For example, a fileStem of "sim" will result in files named sim1.dat, sim2.dat, etc.
miss	Missing data handling template, created by the function miss.
missCode	Missing data code, NA will be replaced by this value for all missing values in exported data.
datafun	Function to be applied to generated data set at each replication.
pmMCAR	The percent completely missing at random. This argument is not necessary except the user wish to vary percent missing completely at random across replications. The pmMCAR here is a vector of percent missing, which the values can be in between 0 and 1 only. The specification of objMissing is not needed (but is needed if users wish to specify complex missing value data generation or wish to use multiple imputation).
pmMAR	The percent missing at random. This argument is not necessary except the user wish to vary percent missing at random across replications. The pmMAR here is a vector of percent missing, which the values can be in between 0 and 1 only. The specification of objMissing is not needed (but is needed if users wish to specify complex missing value data generation or wish to use multiple imputation).

facDist
: A SimDataDist object or list of objects for the distribution of factors. If one object is passed, all factors will have the same distribution. Use when `sequential` is `TRUE`.

indDist
: A SimDataDist object or list of objects for a distribution of indicators. If one object is passed, each indicator will have the same distribution. Use when `sequential` is `FALSE`.

errorDist
: An object or list of objects of type `SimDataDist` indicating the distribution of errors. If a single `SimDataDist` is specified, each error will be genrated with that distribution.

sequential
: If `TRUE`, use a sequential method to create data such that the data from factor are generated first and apply to a set of equations to obtain the data of indicators. If `FALSE`, create data directly from model-implied mean and covariance of indicators.

modelBoot
: When specified, a model-based bootstrap is used for data generation. See draw for further information. This argument requires real data to be passed to `realData`.

realData
: A data.frame containing real data. The data generated will follow the distribution of this data set.

maxDraw
: Integer specifying the maximum number of attempts to draw a valid set of parameters (no negative error variance, standardized coefficients over 1).

misfitType
: Character vector indicating the fit measure used to assess the misfit of a set of parameters. Can be "f0", "rmsea", "srmr", or "all".

misfitBounds
: Vector that contains upper and lower bounds of the misfit measure. Sets of parameters drawn that are not within these bounds are rejected.

averageNumMisspec
: If `TRUE`, the provided fit will be divided by the number of misspecified parameters.

optMisfit
: Character vector of either "min" or "max" indicating either maximum or minimum optimized misfit. If not null, the set of parameters out of the number of draws in "optDraws" that has either the maximum or minimum misfit of the given misfit type will be returned.

optDraws
: Number of parameter sets to draw if optMisfit is not null. The set of parameters with the maximum or minimum misfit will be returned.

seed
: Random number seed. Reproducibility across multiple cores or clusters is ensured using R'Lecuyer package.

silent
: If `TRUE`, suppress warnings.

multicore
: Use multiple processors within a computer. Specify as TRUE to use it.

numProc
: Number of processors for using multiple processors. If it is `NULL`, the package will find the maximum number of processors.

params
: If `TRUE`, the parameters from each replication will be returned.

Value

Text files saved to the current working directory. If `program` = "Mplus" one file is output for each replication, and an extra file is output with the names of all saved data sets (this file can be used with the MONTECARLO command in Mplus). If program = "LISREL" one file is output with each replication stacked on top of the next (this file can be used with the RP command in LISREL). If program = TRUE, a list of parameter values for each replication is returned.

Author(s)

Alexander M. Schoemann (East Carolina University; <schoemanna@ecu.edu>)

Examples

```
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
LY <- bind(loading, 0.7)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))

VY <- bind(rep(NA,6),2)

CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType = "CFA")

## Export 20 replications to an external data file (not run).
#exportData(20, CFA.Model, 200)
```

---

findCoverage — *Find a value of independent variables that provides a given value of coverage rate*

Description

Find a value of independent variable that provides a given value of coverage rate. If there are more than one varying parameters, this function will find the value of the target varying parameters given the values of the other varying parameters.

Usage

```
findCoverage(coverTable, iv, target)
```

Arguments

coverTable	A data.frame providing varying parameters and coverage rates of each parameter. This table is obtained by getPower or continuousPower function.
iv	The target varying parameter that users would like to find the value providing a given power from. This argument can be specified as the index of the target column or the name of target column (i.e., "iv.N" or "N")
target	The target coverage rate

Value

There are five possible types of values provided:

- *Value* The varying parameter value that provides the coverage rate just under the specified coverage rate (the adjacent value of varying parameter provides over power than the specified power value).
- *Minimum value* The minimum value has already provided the low coverage rate (way under the specified coverage rate). The value of varying parameters that provides exact coverage rate may be lower than the minimum value. The example of varying parameter that can provides the minimum value is sample size.
- *Maximum value* The maximum value has already provided the low coverage rate (way under the specified coverage rate). The value of varying parameters that provides exact desired power may be higher than the maximum value. The example of varying parameter that can provides the maximum value is percent missing.
- `NA` There is no value in the domain of varying parameters that provides the coverage rate lower than the desired coverage rate.
- `Inf` The coverage rate of all values in the varying parameters is 0 (specifically more than 0.0001) and any values of the varying parameters can be picked and still provide enough power.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `getCoverage` to find the coverage rate of parameter estimates
- `continuousCoverage` to find the coverage rate of parameter estimates for the result object (`linkS4class{SimResult}`) with varying parameters.

Examples

```
## Not run:
# Specify Sample Size by n
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, 0.4)
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# Specify both sample size and percent missing completely at random. Note that more fine-grained
# values of n and pmMCAR is needed, e.g., n=seq(50, 500, 1) and pmMCAR=seq(0, 0.2, 0.01)
Output <- sim(NULL, model=CFA.Model, n=seq(100, 200, 20), pmMCAR=c(0, 0.1, 0.2))

# Find the power of all possible combination of N and pmMCAR
cover <- getCoverage(Output, coverValue = 0)

# Find the sample size that provides the power of 0.8
```

```
findCoverage(cover, "N", 0.20)

## End(Not run)
```

`findFactorIntercept` *Find factor intercept from regression coefficient matrix and factor total means*

Description

Find factor intercept from regression coefficient matrix and factor total means for latent variable models. In the path analysis model, this function will find indicator intercept from regression coefficient and indicator total means.

Usage

```
findFactorIntercept(beta, factorMean = NULL, gamma = NULL, covmean = NULL)
```

Arguments

`beta`	Regression coefficient matrix among factors
`factorMean`	A vector of total (model-implied) factor (indicator) means. The default is that all total factor means are 0.
`gamma`	Regression coefficient matrix from covariates (column) to factors (rows)
`covmean`	A vector of covariate means.

Value

A vector of factor (indicator) intercepts

Author(s)

Sunthud Pornprasertmanit (`<psunthud@gmail.com>`)

See Also

- `findIndIntercept` to find indicator (measurement) intercepts
- `findIndMean` to find indicator (measurement) total means
- `findIndResidualVar` to find indicator (measurement) residual variances
- `findIndTotalVar` to find indicator (measurement) total variances
- `findFactorMean` to find factor means
- `findFactorResidualVar` to find factor residual variances
- `findFactorTotalVar` to find factor total variances
- `findFactorTotalCov` to find factor covariances

Examples

```
path <- matrix(0, 9, 9)
path[4, 1] <- path[7, 4] <- 0.6
path[5, 2] <- path[8, 5] <- 0.6
path[6, 3] <- path[9, 6] <- 0.6
path[5, 1] <- path[8, 4] <- 0.4
path[6, 2] <- path[9, 5] <- 0.4
factorMean <- c(5, 2, 3, 0, 0, 0, 0, 0, 0)
findFactorIntercept(path, factorMean)
```

---

findFactorMean — *Find factor total means from regression coefficient matrix and factor intercept*

Description

Find factor total means from regression coefficient matrix and factor intercepts for latent variable models. In the path analysis model, this function will find indicator total means from regression coefficient and indicator intercept.

Usage

```
findFactorMean(beta, alpha = NULL, gamma = NULL, covmean = NULL)
```

Arguments

`beta`	Regression coefficient matrix among factors
`alpha`	Factor (indicator) intercept. The default is that all factor intercepts are 0.
`gamma`	Regression coefficient matrix from covariates (column) to factors (rows)
`covmean`	A vector of covariate means.

Value

A vector of factor (indicator) total means

Author(s)

Sunthud Pornprasertmanit (<`psunthud@gmail.com`>)

See Also

- `findIndIntercept` to find indicator (measurement) intercepts
- `findIndMean` to find indicator (measurement) total means
- `findIndResidualVar` to find indicator (measurement) residual variances
- `findIndTotalVar` to find indicator (measurement) total variances
- `findFactorIntercept` to find factor intercepts

- `findFactorResidualVar` to find factor residual variances
- `findFactorTotalVar` to find factor total variances
- `findFactorTotalCov` to find factor covariances

Examples

```
path <- matrix(0, 9, 9)
path[4, 1] <- path[7, 4] <- 0.6
path[5, 2] <- path[8, 5] <- 0.6
path[6, 3] <- path[9, 6] <- 0.6
path[5, 1] <- path[8, 4] <- 0.4
path[6, 2] <- path[9, 5] <- 0.4
intcept <- c(5, 2, 3, 0, 0, 0, 0, 0, 0)
findFactorMean(path, intcept)
```

`findFactorResidualVar` *Find factor residual variances from regression coefficient matrix, factor (residual) correlations, and total factor variances*

Description

Find factor residual variances from regression coefficient matrix, factor (residual) correlation matrix, and total factor variances for latent variable models. In the path analysis model, this function will find indicator residual variances from regression coefficient, indicator (residual) correlation matrix, and total indicator variances.

Usage

```
findFactorResidualVar(beta, corPsi, totalVarPsi = NULL, gamma = NULL, covcov = NULL)
```

Arguments

`beta`	Regression coefficient matrix among factors
`corPsi`	Factor or indicator residual correlations.
`totalVarPsi`	Factor or indicator total variances. The default is that all factor or indicator total variances are 1.
`gamma`	Regression coefficient matrix from covariates (column) to factors (rows)
`covcov`	A covariance matrix among covariates

Value

A vector of factor (indicator) residual variances

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `findIndIntercept` to find indicator (measurement) intercepts
- `findIndMean` to find indicator (measurement) total means
- `findIndResidualVar` to find indicator (measurement) residual variances
- `findIndTotalVar` to find indicator (measurement) total variances
- `findFactorIntercept` to find factor intercepts
- `findFactorMean` to find factor means
- `findFactorTotalVar` to find factor total variances
- `findFactorTotalCov` to find factor covariances

Examples

```
path <- matrix(0, 9, 9)
path[4, 1] <- path[7, 4] <- 0.6
path[5, 2] <- path[8, 5] <- 0.6
path[6, 3] <- path[9, 6] <- 0.6
path[5, 1] <- path[8, 4] <- 0.4
path[6, 2] <- path[9, 5] <- 0.4
facCor <- diag(9)
facCor[1, 2] <- facCor[2, 1] <- 0.4
facCor[1, 3] <- facCor[3, 1] <- 0.4
facCor[2, 3] <- facCor[3, 2] <- 0.4
totalVar <- rep(1, 9)
findFactorResidualVar(path, facCor, totalVar)
```

`findFactorTotalCov` *Find factor total covariance from regression coefficient matrix, factor residual covariance*

Description

Find factor total covariances from regression coefficient matrix, factor residual covariance matrix. The residual covaraince matrix might be derived from factor residual correlation, total variance, and error variance. This function can be applied for path analysis model as well.

Usage

```
findFactorTotalCov(beta, psi = NULL, corPsi = NULL, totalVarPsi = NULL,
    errorVarPsi = NULL, gamma = NULL, covcov = NULL)
```

Arguments

beta	Regression coefficient matrix among factors
psi	Factor or indicator residual covariances. This argument can be skipped if factor residual correlation and either total variances or error variances are specified.
corPsi	Factor or indicator residual correlation. This argument must be specified with total variances or error variances.
totalVarPsi	Factor or indicator total variances.
errorVarPsi	Factor or indicator residual variances.
gamma	Regression coefficient matrix from covariates (column) to factors (rows)
covcov	A covariance matrix among covariates

Value

A matrix of factor (model-implied) total covariance

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- findIndIntercept to find indicator (measurement) intercepts
- findIndMean to find indicator (measurement) total means
- findIndResidualVar to find indicator (measurement) residual variances
- findIndTotalVar to find indicator (measurement) total variances
- findFactorIntercept to find factor intercepts
- findFactorMean to find factor means
- findFactorResidualVar to find factor residual variances
- findFactorTotalVar to find factor total variances

Examples

```
path <- matrix(0, 9, 9)
path[4, 1] <- path[7, 4] <- 0.6
path[5, 2] <- path[8, 5] <- 0.6
path[6, 3] <- path[9, 6] <- 0.6
path[5, 1] <- path[8, 4] <- 0.4
path[6, 2] <- path[9, 5] <- 0.4
facCor <- diag(9)
facCor[1, 2] <- facCor[2, 1] <- 0.4
facCor[1, 3] <- facCor[3, 1] <- 0.4
facCor[2, 3] <- facCor[3, 2] <- 0.4
residualVar <- c(1, 1, 1, 0.64, 0.288, 0.288, 0.64, 0.29568, 0.21888)
findFactorTotalCov(path, corPsi=facCor, errorVarPsi=residualVar)
```

`findFactorTotalVar` *Find factor total variances from regression coefficient matrix, factor (residual) correlations, and factor residual variances*

Description

Find factor total variances from regression coefficient matrix, factor (residual) correlation matrix, and factor residual variances for latent variable models. In the path analysis model, this function will find indicator total variances from regression coefficient, indicator (residual) correlation matrix, and indicator residual variances.

Usage

```
findFactorTotalVar(beta, corPsi, residualVarPsi, gamma = NULL, covcov = NULL)
```

Arguments

`beta`	Regression coefficient matrix among factors
`corPsi`	Factor or indicator residual correlations.
`residualVarPsi`	Factor or indicator residual variances.
`gamma`	Regression coefficient matrix from covariates (column) to factors (rows)
`covcov`	A covariance matrix among covariates

Value

A vector of factor (indicator) total variances

Author(s)

Sunthud Pornprasertmanit (`<psunthud@gmail.com>`)

See Also

- `findIndIntercept` to find indicator (measurement) intercepts
- `findIndMean` to find indicator (measurement) total means
- `findIndResidualVar` to find indicator (measurement) residual variances
- `findIndTotalVar` to find indicator (measurement) total variances
- `findFactorIntercept` to find factor intercepts
- `findFactorMean` to find factor means
- `findFactorResidualVar` to find factor residual variances
- `findFactorTotalCov` to find factor covariances

Examples

```
path <- matrix(0, 9, 9)
path[4, 1] <- path[7, 4] <- 0.6
path[5, 2] <- path[8, 5] <- 0.6
path[6, 3] <- path[9, 6] <- 0.6
path[5, 1] <- path[8, 4] <- 0.4
path[6, 2] <- path[9, 5] <- 0.4
facCor <- diag(9)
facCor[1, 2] <- facCor[2, 1] <- 0.4
facCor[1, 3] <- facCor[3, 1] <- 0.4
facCor[2, 3] <- facCor[3, 2] <- 0.4
residualVar <- c(1, 1, 1, 0.64, 0.288, 0.288, 0.64, 0.29568, 0.21888)
findFactorTotalVar(path, facCor, residualVar)
```

---

findIndIntercept — *Find indicator intercepts from factor loading matrix, total factor mean, and indicator mean.*

Description

Find indicator (measurement) intercepts from a factor loading matrix, total factor mean, and indicator mean.

Usage

```
findIndIntercept(lambda, factorMean = NULL, indicatorMean = NULL,
kappa = NULL, covmean = NULL)
```

Arguments

`lambda`	Factor loading matrix
`factorMean`	Total (model-implied) mean of factors. As a default, all total factor means are 0.
`indicatorMean`	Total indicator means. As a default, all total indicator means are 0.
`kappa`	Regression coefficient matrix from covariates (column) to indicators (rows)
`covmean`	A vector of covariate means.

Value

A vector of indicator (measurement) intercepts.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- findIndMean to find indicator (measurement) total means
- findIndResidualVar to find indicator (measurement) residual variances
- findIndTotalVar to find indicator (measurement) total variances
- findFactorIntercept to find factor intercepts
- findFactorMean to find factor means
- findFactorResidualVar to find factor residual variances
- findFactorTotalVar to find factor total variances
- findFactorTotalCov to find factor covariances

Examples

```
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- c(0.6, 0.7, 0.8)
loading[4:6, 2] <- c(0.6, 0.7, 0.8)
facMean <- c(0.5, 0.2)
indMean <- rep(1, 6)
findIndIntercept(loading, facMean, indMean)
```

---

findIndMean — *Find indicator total means from factor loading matrix, total factor mean, and indicator intercept.*

Description

Find indicator total means from a factor loading matrix, total factor means, and indicator (measurement) intercepts.

Usage

```
findIndMean(lambda, factorMean = NULL, tau = NULL, kappa = NULL,
covmean = NULL)
```

Arguments

lambda	Factor loading matrix
factorMean	Total (model-implied) mean of factors. As a default, all total factor means are 0.
tau	Indicator (measurement) intercepts. As a default, all intercepts are 0.
kappa	Regression coefficient matrix from covariates (column) to indicators (rows)
covmean	A vector of covariate means.

Value

A vector of indicator total means.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- findIndIntercept to find indicator (measurement) intercepts
- findIndResidualVar to find indicator (measurement) residual variances
- findIndTotalVar to find indicator (measurement) total variances
- findFactorIntercept to find factor intercepts
- findFactorMean to find factor means
- findFactorResidualVar to find factor residual variances
- findFactorTotalVar to find factor total variances
- findFactorTotalCov to find factor covariances

Examples

```
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- c(0.6, 0.7, 0.8)
loading[4:6, 2] <- c(0.6, 0.7, 0.8)
facMean <- c(0.5, 0.2)
intcept <- rep(0, 6)
findIndMean(loading, facMean, intcept)
```

---

findIndResidualVar *Find indicator residual variances from factor loading matrix, total factor covariance, and total indicator variances.*

Description

Find indicator (measurement) residual variances from a factor loading matrix, total factor covariance matrix, and total indicator variances.

Usage

```
findIndResidualVar(lambda, totalFactorCov, totalVarTheta = NULL,
kappa = NULL, covcov = NULL)
```

Arguments

lambda	Factor loading matrix
totalFactorCov	Total (model-implied) covariance matrix among factors.
totalVarTheta	Indicator total variances. As a default, all total variances are 1.
kappa	Regression coefficient matrix from covariates (column) to indicators (rows)
covcov	A covariance matrix among covariates

Value

A vector of indicator residual variances.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `findIndIntercept` to find indicator (measurement) intercepts
- `findIndMean` to find indicator (measurement) total means
- `findIndTotalVar` to find indicator (measurement) total variances
- `findFactorIntercept` to find factor intercepts
- `findFactorMean` to find factor means
- `findFactorResidualVar` to find factor residual variances
- `findFactorTotalVar` to find factor total variances
- `findFactorTotalCov` to find factor covariances

Examples

```
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- c(0.6, 0.7, 0.8)
loading[4:6, 2] <- c(0.6, 0.7, 0.8)
facCov <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
totalVar <- rep(1, 6)
findIndResidualVar(loading, facCov, totalVar)
```

`findIndTotalVar` *Find indicator total variances from factor loading matrix, total factor covariance, and indicator residual variances.*

Description

Find indicator total variances from a factor loading matrix, total factor covariance matrix, and indicator (measurement) residual variances.

Usage

```
findIndTotalVar(lambda, totalFactorCov, residualVarTheta, kappa = NULL,
covcov = NULL)
```

Arguments

`lambda`	Factor loading matrix
`totalFactorCov`	Total (model-implied) covariance matrix among factors.
`residualVarTheta`	Indicator (measurement) residual variances.
`kappa`	Regression coefficient matrix from covariates (column) to indicators (rows)
`covcov`	A covariance matrix among covariates

Value

A vector of indicator total variances.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `findIndIntercept` to find indicator (measurement) intercepts
- `findIndMean` to find indicator (measurement) total means
- `findIndResidualVar` to find indicator (measurement) residual variances
- `findFactorIntercept` to find factor intercepts
- `findFactorMean` to find factor means
- `findFactorResidualVar` to find factor residual variances
- `findFactorTotalVar` to find factor total variances
- `findFactorTotalCov` to find factor covariances

Examples

```
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- c(0.6, 0.7, 0.8)
loading[4:6, 2] <- c(0.6, 0.7, 0.8)
facCov <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
resVar <- c(0.64, 0.51, 0.36, 0.64, 0.51, 0.36)
findIndTotalVar(loading, facCov, resVar)
```

`findPossibleFactorCor` *Find the appropriate position for freely estimated correlation (or covariance) given a regression coefficient matrix*

Description

Find the appropriate position for freely estimated correlation (or covariance) given a regression coefficient matrix. The appropriate position is the pair of variables that are not causally related.

Usage

```
findPossibleFactorCor(beta)
```

Arguments

`beta`	The regression coefficient in path analysis.

Value

The symmetric matrix containing the appropriate position for freely estimated correlation.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `findRecursiveSet` to group variables regarding the position in mediation chain.

Examples

```
path <- matrix(0, 9, 9)
path[4, 1] <- path[7, 4] <- NA
path[5, 2] <- path[8, 5] <- NA
path[6, 3] <- path[9, 6] <- NA
path[5, 1] <- path[8, 4] <- NA
path[6, 2] <- path[9, 5] <- NA
findPossibleFactorCor(path)
```

`findPower` *Find a value of independent variables that provides a given value of power.*

Description

Find a value of independent variable that provides a given value of power. If there are more than one varying parameters, this function will find the value of the target varying parameters given the values of the other varying parameters.

Usage

```
findPower(powerTable, iv, power)
```

Arguments

`powerTable`	A `data.frame` providing varying parameters and powers of each parameter. This table is obtained by `getPower` or `continuousPower` function.
`iv`	The target varying parameter that users would like to find the value providing a given power from. This argument can be specified as the index of the target column or the name of target column (i.e., `"iv.N"` or `"N"`)
`power`	A desired power.

Value

There are five possible types of values provided:

- *Value* The varying parameter value that provides the power just over the specified power value (the adjacent value of varying parameter provides lower power than the specified power value).
- *Minimum value* The minimum value has already provided enough power (way over the specified power value). The value of varying parameters that provides exact desired power may be lower than the minimum value. The example of varying parameter that can provides the minimum value is sample size.
- *Maximum value* The maximum value has already provided enough power (way over the specified power value). The value of varying parameters that provides exact desired power may be higher than the maximum value. The example of varying parameter that can provides the maximum value is percent missing.
- `NA` There is no value in the domain of varying parameters that provides the power greater than the desired power.
- `Inf` The power of all values in the varying parameters is 1 (specifically more than 0.9999) and any values of the varying parameters can be picked and still provide enough power.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- getPower to find the power of parameter estimates
- continuousPower to find the power of parameter estimates for the result object (linkS4class{SimResult}) with varying parameters.

Examples

```
## Not run:
# Specify Sample Size by n
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, 0.4)
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# Specify both sample size and percent missing completely at random. Note that more fine-grained
# values of n and pmMCAR is needed, e.g., n=seq(50, 500, 1) and pmMCAR=seq(0, 0.2, 0.01)
Output <- sim(NULL, model=CFA.Model, n=seq(100, 200, 20), pmMCAR=c(0, 0.1, 0.2))

# Find the power of all possible combination of N and pmMCAR
pow <- getPower(Output)

# Find the sample size that provides the power of 0.8
findPower(pow, "N", 0.80)

## End(Not run)
```

---

findRecursiveSet *Group variables regarding the position in mediation chain*

Description

In mediation analysis, variables affects other variables as a chain. This function will group variables regarding the chain of mediation analysis.

Usage

```
findRecursiveSet(beta)
```

Arguments

beta	The regression coefficient in path analysis.

Value

The list of set of variables in the mediation chain. The variables in position 1 will be the independent variables. The variables in the last variables will be the end of the chain.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `findPossibleFactorCor` to find the possible position for latent correlation given a regression coefficient matrix

Examples

```
path <- matrix(0, 9, 9)
path[4, 1] <- path[7, 4] <- NA
path[5, 2] <- path[8, 5] <- NA
path[6, 3] <- path[9, 6] <- NA
path[5, 1] <- path[8, 4] <- NA
path[6, 2] <- path[9, 5] <- NA
findRecursiveSet(path)
```

`generate` *Generate data using SimSem template*

Description

This function can be used to generate random data based on the 1. `SimSem` objects created with the `model` function, 2. `lavaan` script or parameter tables, or 3. an `MxModel` object from the `OpenMx` package. Some notable features include fine control of misspecification and misspecification optimization (for `SimSem` only), as well as the ability to generate non-normal data. When using *simsem* for simulations, this function is used internally to generate data in the function `sim`, and can be helpful for debugging, or in creating data for use with other analysis programs.

Usage

```
generate(model, n, maxDraw=50, misfitBounds=NULL, misfitType="f0",
averageNumMisspec=FALSE, optMisfit=NULL, optDraws=50,
createOrder = c(1, 2, 3), indDist=NULL, sequential=FALSE,
facDist=NULL, errorDist=NULL, saveLatentVar = FALSE, indLab=NULL,
modelBoot=FALSE, realData=NULL, covData=NULL, params=FALSE, group = NULL,
empirical = FALSE, ...)
```

Arguments

`model`	A `SimSem` object, a `lavaan` script or parameter tables, or an `MxModel` object from the `OpenMx` package
`n`	Integer of sample size.
`maxDraw`	Integer specifying the maximum number of attempts to draw a valid set of parameters (no negative error variance, standardized coefficients over 1).

misfitBounds
: Vector that contains upper and lower bounds of the misfit measure. Sets of parameters drawn that are not within these bounds are rejected.

misfitType
: Character vector indicating the fit measure used to assess the misfit of a set of parameters. Can be "f0", "rmsea", "srmr", or "all".

averageNumMisspec
: If `TRUE`, the provided fit will be divided by the number of misspecified parameters.

optMisfit
: Character vector of either "min" or "max" indicating either maximum or minimum optimized misfit. If not null, the set of parameters out of the number of draws in "optDraws" that has either the maximum or minimum misfit of the given misfit type will be returned.

optDraws
: Number of parameter sets to draw if optMisfit is not null. The set of parameters with the maximum or minimum misfit will be returned.

createOrder
: The order of 1) applying equality/inequality constraints, 2) applying misspecification, and 3) fill unspecified parameters (e.g., residual variances when total variances are specified). The specification of this argument is a vector of different orders of 1 (constraint), 2 (misspecification), and 3 (filling parameters). For example, `c(1,2,3)` is to apply constraints first, then add the misspecification, and finally fill all parameters. See the example of how to use it in the `draw` function.

indDist
: A `SimDataDist` object or list of objects for a distribution of indicators. If one object is passed, each indicator will have the same distribution. Use when `sequential` is `FALSE`.

sequential
: If `TRUE`, use a sequential method to create data such that the data from factor are generated first and apply to a set of equations to obtain the data of indicators. If `FALSE`, create data directly from model-implied mean and covariance of indicators.

facDist
: A `SimDataDist` object or list of objects for the distribution of factors. If one object is passed, all factors will have the same distribution. Use when `sequential` is `TRUE`.

errorDist
: An object or list of objects of type `SimDataDist` indicating the distribution of errors. If a single `SimDataDist` is specified, each error will be genrated with that distribution.

saveLatentVar
: If `TRUE`, the total latent variable scores, residual latent variable scores, and measurement error scores are also provided as the `"latentVar"` attribute of the generated data by the following line: `attr(generatedData,"latentVar")`. The `sequential` argument must be `TRUE` in order to use this option.

indLab
: A vector of indicator labels. When not specified, the variable names are `x1,x2,... xN`.

modelBoot
: When specified, a model-based bootstrap is used for data generation. See details for further information. This argument requires real data to be passed to `realData`.

realData
: A data.frame containing real data. The data generated will follow the distribution of this data set.

covData	A data.frame containing covariate data, which can have any distributions. This argument is required when users specify GA or KA matrices in the model template (SimSem).
params	If TRUE, return the parameters drawn along with the generated data set. Default is FALSE.
group	The label of the grouping variable
empirical	Logical. If TRUE, the specified parameters are treated as sample statistics and data are created to get the specified sample statistics. This argument is applicable when multivariate normal distribution is specified only.
...	Additional arguments for the simulateData function.

Details

If the lavaan script or the MxModel are provided, the model-implied covariance matrix will be computed and internally use createData function to generate data. The data-generation method is based on whether the indDist argument is specified. For the lavaan script, the code for data generation is modified from the simulateData function.

If the SimSem object is specified, it will check whether there are any random parameters or trivial misspecification in the model. If so, real or misspecified parameters are drawn via the draw function. Next, there are two methods to generate data. First, the function will calculate the model-implied covariance matrix (including model misspecification) and generate data similar to the lavaan script or the MxModel object. The second method is referred to as the sequential method, which can be used by specifying the sequential argument as TRUE. This function will create data based on the chain of equations in structural equation modeling such that independent variables and errors are generated and added as dependent variables and the dependent variables will be treated as independent variables in the next equation. For example, in the model with factor A and B are independent variables, factor C are dependent variables, factors A and B are generated first. Then, residual in factor C are created and added with factors A and B. This current step has all factor scores. Then, measurement errors are created and added with factor scores to create indicator scores. During each step, independent variables and errors can be nonnormal by setting facDist or errorDist arguments. The data generation in each step is based on the createData function.

For the model-based bootstrap (providing the realData argument), the transformation proposed by Yung & Bentler (1996) is used. This procedure is the expansion from the Bollen and Stine (1992) bootstrap including a mean structure. The model-implied mean vector and covariance matrix with trivial misspecification will be used in the model-based bootstrap if misspec is specified. See page 133 of Bollen and Stine (1992) for a reference.

Value

A data.frame containing simulated data from the data generation template. A variable "group" is appended indicating group membership.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>), Patrick Miller (University of Notre Dame; <pmille13@nd.edu>), the data generation code for lavaan script is modifed from the simulateData function in lavaan written by Yves Rosseel

References

Bollen, K. A., & Stine, R. A. (1992). Bootstrapping goodness-of-fit measures in structural equation models. *Sociological Methods and Research, 21,* 205-229.

Yung, Y.-F., & Bentler, P. M. (1996). Bootstrapping techniques in analysis of mean and covariance structures. In G. A. Marcoulides & R. E. Schumacker (Eds.), *Advanced structural equation modeling: Issues and techniques* (pp. 195-226). Mahwah, NJ: Erlbaum.

See Also

- `draw` To draw parameters using the `SimSem` template.
- `createData` To generate random data using a set of parameters from `draw`

Examples

```
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
LY <- bind(loading, 0.7)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))

VY <- bind(rep(NA,6),2)

CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType = "CFA")

dat <- generate(CFA.Model, 200)

# Get the latent variable scores

dat2 <- generate(CFA.Model, 20, sequential = TRUE, saveLatentVar = TRUE)
dat2
attr(dat2, "latentVar")
```

`getCIwidth` *Find confidence interval width*

Description

Find the median of confidence interval width or a confidence interval value given a degree of assurance (Lai & Kelley, 2011)

Usage

```
getCIwidth(object, assurance = 0.50, nVal = NULL, pmMCARval = NULL,
pmMARval = NULL, df = 0)
```

Arguments

- `object` — `SimResult` that saves the analysis results from multiple replications
- `assurance` — The percentile of the resulting confidence interval width. When assurance is 0.50, the median of the widths is provided. See Lai & Kelley (2011) for more details.
- `nVal` — The sample size value that researchers wish to find the confidence interval width from. This argument is applicable for `SimResult` with varying sample sizes or percent missing across replications
- `pmMCARval` — The percent missing completely at random value that researchers wish to find the confidence interval width from. This argument is applicable for `SimResult` with varying sample sizes or percent missing across replications
- `pmMARval` — The percent missing at random value that researchers wish to find the confidence interval width from. This argument is applicable for `SimResult` with varying sample sizes or percent missing across replications
- `df` — The degree of freedom used in spline method in predicting the confidence interval width by the predictors. If `df` is 0, the spline method will not be applied. This argument is applicable for `SimResult` with varying sample sizes or percent missing across replications

Value

The median of confidence interval width or a confidence interval given a degree of assurance

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

Lai, K., & Kelley, K. (2011). Accuracy in parameter estimation for targeted effects in structural equation modeling: Sample size planning for narrow confidence intervals. *Psychological Methods, 16*, 127-148.

See Also

`SimResult` for a detail of simResult

Examples

```
## Not run:
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loadingValues <- matrix(0, 6, 2)
loadingValues[1:3, 1] <- 0.7
loadingValues[4:6, 2] <- 0.7
LY <- bind(loading, loadingValues)
latent.cor <- matrix(NA, 2, 2)
```

```
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTE <- binds(error.cor)
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output <- sim(5, n = 200, model=CFA.Model)

# Get the cutoff (critical value) when alpha is 0.05
getCIwidth(Output, assurance=0.80)

# Finding the cutoff when the sample size is varied. Note that more fine-grained
# values of n is needed, e.g., n=seq(50, 500, 1)
Output2 <- sim(NULL, model=CFA.Model, n=seq(50, 100, 10))

# Get the fit index cutoff when sample size is 75.
getCIwidth(Output2, assurance=0.80, nVal = 75)

## End(Not run)
```

`getCoverage` *Find coverage rate of model parameters*

Description

A function to find the coverage rate of confidence intervals in a model when none, one, or more of the simulations parameters vary randomly across replications.

Usage

```
getCoverage(simResult, coverValue = NULL, contParam = NULL, coverParam = NULL,
    nVal = NULL, pmMCARval = NULL, pmMARval = NULL, paramVal = NULL)
```

Arguments

`simResult`	`SimResult` that may include randomly varying parameters (e.g. sample size, percent missing, model parameters)
`coverValue`	A target value used that users wish to find the coverage rate of that value (e.g., 0). If `NULL`, the parameter values will be used.
`contParam`	Vector of parameters names that vary over replications.
`coverParam`	Vector of parameters names that the user wishes to find coverage rate for. This can be a vector of names (e.g., "f1=~y2", "f1~~f2"). If parameters are not specified, coverage rates for all parameters in the model will be returned.
`nVal`	The sample size values that users wish to find power from.

pmMCARval The percent completely missing at random values that users wish to find power from.

pmMARval The percent missing at random values that users wish to find power from.

paramVal A list of varying parameter values that users wish to find power from.

Details

In this function, the coverage (which can be 0 or 1) is regressed on randomly varying simulation parameters (e.g., sample size, percentage of missing data, or model parameters) using logistic regression. For a set of independent variables values, the predicted probability from the logistic regression equation is the predicted coverage rate.

Value

Data frame containing columns representing values of the randomly varying simulation parameters, and coverage rates for model parameters of interest.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>), Alexander M. Schoemann (East Carolina University; <schoemanna@ecu.edu>)

See Also

- SimResult to see how to create a simResult object with randomly varying parameters.

Examples

```
## Not run:
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, 0.7)
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# Specify both sample size and percent missing completely at random. Note that more fine-grained
# values of n and pmMCAR is needed, e.g., n=seq(50, 500, 1) and pmMCAR=seq(0, 0.2, 0.01)
Output <- sim(NULL, model=CFA.Model, n=seq(100, 200, 20), pmMCAR=c(0, 0.1, 0.2))
summary(Output)

# Get the coverage rates of all possible combinations of n and pmMCAR
getCoverage(Output)

# Get the coverage rates of the combinations of n of 100 and 200 and pmMCAR of 0, 0.1, and 0.2
getCoverage(Output, coverValue = 0, nVal=c(100, 200), pmMCARval=c(0, 0.1, 0.2))

## End(Not run)
```

getCutoff *Find fit indices cutoff given a priori alpha level*

Description

Extract fit indices information from the SimResult and get the cutoffs of fit indices given a priori alpha level

Usage

```
getCutoff(object, alpha, revDirec = FALSE, usedFit = NULL, nVal = NULL,
pmMCARval = NULL, pmMARval = NULL, df = 0)
```

Arguments

object	SimResult that saves the analysis results from multiple replications
alpha	A priori alpha level
revDirec	The default is to find criticl point on the side that indicates worse fit (the right side of RMSEA or the left side of CFI). If specifying as TRUE, the directions are reversed.
usedFit	Vector of names of fit indices that researchers wish to getCutoffs from. The default is to getCutoffs of all fit indices.
nVal	The sample size value that researchers wish to find the fit indices cutoffs from. This argument is applicable for SimResult with varying sample sizes or percent missing across replications
pmMCARval	The percent missing completely at random value that researchers wish to find the fit indices cutoffs from. This argument is applicable for SimResult with varying sample sizes or percent missing across replications
pmMARval	The percent missing at random value that researchers wish to find the fit indices cutoffs from. This argument is applicable for SimResult with varying sample sizes or percent missing across replications
df	The degree of freedom used in spline method in predicting the fit indices by the predictors. If df is 0, the spline method will not be applied. This argument is applicable for SimResult with varying sample sizes or percent missing across replications

Value

One-tailed cutoffs of several fit indices with a priori alpha level

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

SimResult for a detail of simResult

Examples

```
## Not run:
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loadingValues <- matrix(0, 6, 2)
loadingValues[1:3, 1] <- 0.7
loadingValues[4:6, 2] <- 0.7
LY <- bind(loading, loadingValues)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTE <- binds(error.cor)
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output <- sim(5, n = 200, model=CFA.Model)

# Get the cutoff (critical value) when alpha is 0.05
getCutoff(Output, 0.05)

# Finding the cutoff when the sample size is varied. Note that more fine-grained
# values of n is needed, e.g., n=seq(50, 500, 1)
Output2 <- sim(NULL, model=CFA.Model, n=seq(50, 100, 10))

# Get the fit index cutoff when sample size is 75.
getCutoff(Output2, 0.05, nVal = 75)

## End(Not run)
```

---

getCutoffNested — *Find fit indices cutoff for nested model comparison given a priori alpha level*

Description

Extract fit indices information from the simulation of parent and nested models and getCutoff of fit indices given a priori alpha level

Usage

```
getCutoffNested(nested, parent, alpha = 0.05, usedFit = NULL, nVal = NULL,
pmMCARval = NULL, pmMARval = NULL, df = 0)
```

Arguments

- `nested` — SimResult that saves the analysis results of nested model from multiple replications
- `parent` — SimResult that saves the analysis results of parent model from multiple replications
- `alpha` — A priori alpha level
- `usedFit` — Vector of names of fit indices that researchers wish to getCutoffs from. The default is to getCutoffs of all fit indices.
- `nVal` — The sample size value that researchers wish to find the fit indices cutoffs from.
- `pmMCARval` — The percent missing completely at random value that researchers wish to find the fit indices cutoffs from.
- `pmMARval` — The percent missing at random value that researchers wish to find the fit indices cutoffs from.
- `df` — The degree of freedom used in spline method in predicting the fit indices by the predictors. If `df` is 0, the spline method will not be applied.

Value

One-tailed cutoffs of several fit indices with a priori alpha level

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

SimResult for a detail of simResult getCutoff for a detail of finding cutoffs for absolute fit

Examples

```
## Not run:
# Nested Model
loading.null <- matrix(0, 6, 1)
loading.null[1:6, 1] <- NA
LY.NULL <- bind(loading.null, 0.7)
RPS.NULL <- binds(diag(1))

error.cor.mis <- matrix("rnorm(1, 0, 0.1)", 6, 6)
diag(error.cor.mis) <- 1
RTE <- binds(diag(6), misspec=error.cor.mis)
CFA.Model.NULL <- model(LY = LY.NULL, RPS = RPS.NULL, RTE = RTE, modelType="CFA")

# Parent Model
loading.alt <- matrix(0, 6, 2)
loading.alt[1:3, 1] <- NA
loading.alt[4:6, 2] <- NA
LY.ALT <- bind(loading.alt, 0.7)
latent.cor.alt <- matrix(NA, 2, 2)
```

```
diag(latent.cor.alt) <- 1
RPS.ALT <- binds(latent.cor.alt, "runif(1, 0.7, 0.9)")
CFA.Model.ALT <- model(LY = LY.ALT, RPS = RPS.ALT, RTE = RTE, modelType="CFA")

# The actual number of replications should be greater than 10.
Output.NULL.NULL <- sim(10, n=500, model=CFA.Model.NULL, generate=CFA.Model.NULL)
Output.NULL.ALT <- sim(10, n=500, model=CFA.Model.ALT, generate=CFA.Model.NULL)

# Find the fix index cutoff from the sampling distribution of the difference
# in fit index of nested models where the alpha is 0.05.
getCutoffNested(Output.NULL.NULL, Output.NULL.ALT, alpha=0.05)

## End(Not run)
```

`getCutoffNonNested` *Find fit indices cutoff for non-nested model comparison given a priori alpha level*

Description

Extract fit indices information from the simulation of two models fitting on the datasets created from both models and getCutoff of fit indices given a priori alpha level

Usage

```
getCutoffNonNested(dat1Mod1, dat1Mod2, dat2Mod1=NULL, dat2Mod2=NULL,
alpha=.05, usedFit=NULL, onetailed=FALSE, nVal = NULL, pmMCARval = NULL,
pmMARval = NULL, df = 0)
```

Arguments

`dat1Mod1`	`SimResult` that saves the simulation of analyzing Model 1 by datasets created from Model 1
`dat1Mod2`	`SimResult` that saves the simulation of analyzing Model 2 by datasets created from Model 1
`dat2Mod1`	`SimResult` that saves the simulation of analyzing Model 1 by datasets created from Model 2
`dat2Mod2`	`SimResult` that saves the simulation of analyzing Model 2 by datasets created from Model 2
`alpha`	A priori alpha level
`usedFit`	Vector of names of fit indices that researchers wish to get cutoffs from. The default is to get cutoffs of all fit indices.
`onetailed`	If `TRUE`, the function will find the cutoff from one-tail test. If `FALSE`, the funciton will find the cutoff from two-tailed test.
`nVal`	The sample size value that researchers wish to find the fit indices cutoffs from.

`pmMCARval` The percent missing completely at random value that researchers wish to find the fit indices cutoffs from.

`pmMARval` The percent missing at random value that researchers wish to find the fit indices cutoffs from.

`df` The degree of freedom used in spline method in predicting the fit indices by the predictors. If `df` is 0, the spline method will not be applied.

Value

One- or two-tailed cutoffs of several fit indices with a priori alpha level. The cutoff is based on the fit indices from Model 1 subtracted by the fit indices from Model 2.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

`SimResult` for a detail of simResult `getCutoff` for a detail of finding cutoffs for absolute fit `getCutoffNested` for a detail of finding cutoffs for nested model comparison `plotCutoffNonNested` Plot cutoffs for non-nested model comparison

Examples

```
## Not run:
# Model A: Factor 1 with items 1-3 and Factor 2 with items 4-8
loading.A <- matrix(0, 8, 2)
loading.A[1:3, 1] <- NA
loading.A[4:8, 2] <- NA
LY.A <- bind(loading.A, 0.7)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, "runif(1, 0.7, 0.9)")
RTE <- binds(diag(8))
CFA.Model.A <- model(LY = LY.A, RPS = RPS, RTE = RTE, modelType="CFA")

# Model B: Factor 1 with items 1-4 and Factor 2 with items 5-8
loading.B <- matrix(0, 8, 2)
loading.B[1:4, 1] <- NA
loading.B[5:8, 2] <- NA
LY.B <- bind(loading.B, 0.7)
CFA.Model.B <- model(LY = LY.B, RPS = RPS, RTE = RTE, modelType="CFA")

# The actual number of replications should be greater than 10.
Output.A.A <- sim(10, n=500, model=CFA.Model.A, generate=CFA.Model.A)
Output.A.B <- sim(10, n=500, model=CFA.Model.B, generate=CFA.Model.A)
Output.B.A <- sim(10, n=500, model=CFA.Model.A, generate=CFA.Model.B)
Output.B.B <- sim(10, n=500, model=CFA.Model.B, generate=CFA.Model.B)

# Find the cutoffs from the sampling distribution to reject model A (model 1)
# and to reject model B (model 2)
```

```
getCutoffNonNested(Output.A.A, Output.A.B, Output.B.A, Output.B.B)

# Find the cutoffs from the sampling distribution to reject model A (model 1)
getCutoffNonNested(Output.A.A, Output.A.B)

# Find the cutoffs from the sampling distribution to reject model B (model 1)
getCutoffNonNested(Output.B.B, Output.B.A)

## End(Not run)
```

`getExtraOutput` *Get extra outputs from the result of simulation*

Description

Get extra outputs from a simulation result object (`SimResult`). Users can ask this package to extra output from the `lavaan` object in each iteration by setting the `outfun` argument (in the `sim` function). See the example below.

Usage

```
getExtraOutput(object, improper = TRUE, nonconverged = FALSE)
```

Arguments

`object`	`SimResult` that have the extra output extracted by the function defined in the `outfun` argument (in the `sim` function)
`improper`	Specify whether to include the information from the replications with improper solutions
`nonconverged`	Specify whether to include the information from the nonconvergent replications

Value

A list of extra outputs

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `sim` A function to run a Monte Carlo simulation

Examples

```
## Not run:
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, 0.7)
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# Write a function to extract the modification index from lavaan object
outfun <- function(out) {
result <- inspect(out, "mi")
}

# We will use only 5 replications to save time.
# In reality, more replications are needed.
Output <- sim(5, n=200, model=CFA.Model, outfun=outfun)

# Get the modification index of each replication
getExtraOutput(Output)

## End(Not run)
```

---

getPopulation — *Extract the data generation population model underlying a result object*

Description

This function will extract the data generation population model underlying a result object (`linkS4class{SimResult}`).

Usage

```
getPopulation(object, std = FALSE, improper = TRUE, nonconverged = FALSE)
```

Arguments

`object`	The result object that you wish to extract the data generation population model from (`linkS4class{SimResult}`).
`std`	If `TRUE`, standardized parameters are returned.
`improper`	Specify whether to include the information from the replications with improper solutions
`nonconverged`	Specify whether to include the information from the nonconvergent replications

Value

A data frame contained the population of each replication

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- SimResult for result object

Examples

```
## Not run:
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, "runif(1, 0.4, 0.9)")
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# We will use only 10 replications to save time.
# In reality, more replications are needed.
Output <- sim(10, n=200, model=CFA.Model)

# Get the population parameters
getPopulation(Output)

## End(Not run)
```

---

getPower — *Find power of model parameters*

Description

A function to find the power of parameters in a model when none, one, or more of the simulations parameters vary randomly across replications.

Usage

```
getPower(simResult, alpha = 0.05, contParam = NULL, powerParam = NULL,
nVal = NULL, pmMCARval = NULL, pmMARval = NULL, paramVal = NULL)
```

Arguments

simResult	SimResult that may include randomly varying parameters (e.g. sample size, percent missing, model parameters)
alpha	Alpha level to use for power analysis.
contParam	Vector of parameters names that vary over replications.
powerParam	Vector of parameters names that the user wishes to find power for. This can be a vector of names (e.g., "f1=~y2", "f1~~f2"). If parameters are not specified, power for all parameters in the model will be returned.

`nVal` The sample size values that users wish to find power from.

`pmMCARval` The percent completely missing at random values that users wish to find power from.

`pmMARval` The percent missing at random values that users wish to find power from.

`paramVal` A list of varying parameter values that users wish to find power from.

Details

A common use of simulations is to conduct power analyses, especially when using SEM (Muthen & Muthen, 2002). Here, researchers could select values for each parameter and a sample size and run a simulation to determine power in those conditions (the proportion of generated datasets in which a particular parameter of interest is significantly different from zero). To evaluate power at multiple sample sizes, one simulation for each sample size must be run. This function not only calculate power for each sample size but also calculate power for multiple sample sizes varying continuously. By continuously varying sample size across replications, only a single simulation is needed. In this simulation, the sample size for each replication varies randomly across plausible sample sizes (e.g., sample sizes between 200 and 500). For each replication, the sample size and significance of each parameter (0 = not significant, 1 = significant) are recorded. When the simulation is complete, parameter significance is regressed on sample size using logistic regression. For a given sample size, the predicted probability from the logistic regression equation is the power to detect an effect at that sample size. This approach can be extended to other randomly varying simulation parameters such as the percentage of missing data, and model parameters.

Value

Data frame containing columns representing values of the randomly varying simulation parameters, and power for model parameters of interest.

Author(s)

Alexander M. Schoemann (East Carolina University; `<schoemanna@ecu.edu>`), Sunthud Pornprasertmanit (`<psunthud@gmail.com>`)

References

Muthen, L. K., & Muthen, B. O. (2002). How to use a Monte Carlo study to decide on sample size and determine power. *Structural Equation Modeling, 4,* 599-620.

See Also

- `SimResult` to see how to create a simResult object with randomly varying parameters.

Examples

```
## Not run:
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, 0.7)
RPS <- binds(diag(1))
```

```
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# Specify both sample size and percent missing completely at random. Note that more fine-grained
# values of n and pmMCAR is needed, e.g., n=seq(50, 500, 1) and pmMCAR=seq(0, 0.2, 0.01)
Output <- sim(NULL, model=CFA.Model, n=seq(100, 200, 20), pmMCAR=c(0, 0.1, 0.2))
summary(Output)

# Get the power of all possible combinations of n and pmMCAR
getPower(Output)

# Get the power of the combinations of n of 100 and 200 and pmMCAR of 0, 0.1, and 0.2
getPower(Output, nVal=c(100, 200), pmMCARval=c(0, 0.1, 0.2))

## End(Not run)
```

---

getPowerFit *Find power in rejecting alternative models based on fit indices criteria*

Description

Find the proportion of fit indices that indicate worse fit than a specified cutoffs. The cutoffs may be calculated from `getCutoff` of the null model.

Usage

```
getPowerFit(altObject, cutoff = NULL, nullObject = NULL, revDirec = FALSE,
usedFit = NULL, alpha = 0.05, nVal = NULL, pmMCARval = NULL, pmMARval = NULL,
condCutoff = TRUE, df = 0)
```

Arguments

altObject	`SimResult` that indicates alternative model that users wish to reject
cutoff	Fit indices cutoffs from null model or users. This should be a vector with a specified fit indices names as the name of vector elements. The `cutoff` cannot be specified if the `nullObject` is specified.
nullObject	The `SimResult` that contains the simulation result from fitting the null model by the data from the null model. The `nullObject` cannot be specified if the `cutoff` is specified.
revDirec	Reverse the direction of deciding a power by fit indices (e.g., less than –> greater than). The default is to count the proportion of fit indices that indicates lower fit to the model, such as how many RMSEA in the alternative model that is worse than cutoffs. The direction can be reversed by setting as `TRUE`.
usedFit	The vector of names of fit indices that researchers wish to get powers from. The default is to get powers of all fit indices
alpha	The alpha level used to find the cutoff if the `nullObject` is specified. This argument is not applicable if the `cutoff` is specified.

nVal	The sample size value that researchers wish to find the power from. This argument is applicable when altObject has a random sample size.
pmMCARval	The percent missing completely at random value that researchers wish to find the power from. This argument is applicable when altObject has a random percent missing completely at random.
pmMARval	The percent missing at random value that researchers wish to find the power from. This argument is applicable when altObject has a random percent missing at random.
condCutoff	A logical value to use a conditional quantile method (if TRUE) or logistic regression method (if FALSE) to find the power. The conditional quantile method use quantile regression to find the quantile of the cutoff on the alternative sampling distribution that varies nVal, pmMCARval, or pmMARval. The value of 1 - quantile will be reported as the power given the set of nVal, pmMCARval, and pmMARval. The logistic regression method is based on transforming the fit indices value to reject/retain decision first. Then, the logistic regression is used to predict reject/retain decision given the set of nVal, pmMCARval, and pmMARval. The predicted probability is reported as a power. This argument is applicable for specification of cutoff only.
df	The degree of freedom used in spline method in quantile regression (condCutoff = TRUE). If df is 0, which is the default, the spline method will not be applied.

Value

List of power given different fit indices. The TraditionalChi means the proportion of replications that are rejected by the traditional chi-square test.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- getCutoff to find the cutoffs from null model.
- SimResult to see how to create simResult

Examples

```
## Not run:
# Null model with one factor
loading.null <- matrix(0, 6, 1)
loading.null[1:6, 1] <- NA
LY.NULL <- bind(loading.null, 0.7)
RPS.NULL <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model.NULL <- model(LY = LY.NULL, RPS = RPS.NULL, RTE = RTE, modelType="CFA")

# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output.NULL <- sim(5, n=500, model=CFA.Model.NULL)
```

```
# Get the fit index cutoff from the null model
Cut.NULL <- getCutoff(Output.NULL, 0.05)

# Alternative model with two factor
loading.alt <- matrix(0, 6, 2)
loading.alt[1:3, 1] <- NA
loading.alt[4:6, 2] <- NA
LY.ALT <- bind(loading.alt, 0.7)
latent.cor.alt <- matrix(NA, 2, 2)
diag(latent.cor.alt) <- 1
RPS.ALT <- binds(latent.cor.alt, "runif(1, 0.7, 0.9)")
CFA.Model.ALT <- model(LY = LY.ALT, RPS = RPS.ALT, RTE = RTE, modelType="CFA")

# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output.ALT <- sim(5, n=500, model=CFA.Model.NULL, generate=CFA.Model.ALT)

# Get the power based on the derived cutoff
getPowerFit(Output.ALT, cutoff=Cut.NULL)

# Get the power based on the rule of thumb proposed by Hu & Bentler (1999)
Rule.of.thumb <- c(RMSEA=0.05, CFI=0.95, TLI=0.95, SRMR=0.06)
getPowerFit(Output.ALT, cutoff=Rule.of.thumb, usedFit=c("RMSEA", "CFI", "TLI", "SRMR"))

# The example of continous varying sample size. Note that more fine-grained
# values of n is needed, e.g., n=seq(50, 500, 1)
Output.NULL2 <- sim(NULL, n=seq(50, 500, 50), model=CFA.Model.NULL, generate=CFA.Model.NULL)
Output.ALT2 <- sim(NULL, n=seq(50, 500, 50), model=CFA.Model.NULL, generate=CFA.Model.ALT)

# Get the power based on the derived cutoff from the null model at the sample size of 250
getPowerFit(Output.ALT2, nullObject=Output.NULL2, nVal=250)

## End(Not run)
```

`getPowerFitNested` *Find power in rejecting nested models based on the differences in fit indices*

Description

Find the proportion of the difference in fit indices that indicate worse fit than a specified (or internally derived) cutoffs.

Usage

```
getPowerFitNested(altNested, altParent, cutoff = NULL, nullNested = NULL,
nullParent = NULL, revDirec = FALSE, usedFit = NULL, alpha = 0.05, nVal = NULL,
pmMCARval = NULL, pmMARval = NULL, condCutoff = TRUE, df = 0)
```

Arguments

altNested
: SimResult that saves the simulation result of the nested model when the nested model is FALSE.

altParent
: SimResult that saves the simulation result of the parent model when the nested model is FALSE.

cutoff
: A vector of priori cutoffs for fit indices. The cutoff cannot be specified if the nullNested and nullParent are specified.

nullNested
: The SimResult that saves the simulation result of the nested model when the nested model is TRUE. This argument must be specified with nullParent. The nullNested cannot be specified if the cutoff is specified.

nullParent
: The SimResult that saves the simulation result of the parent model when the nested model is TRUE. This argument must be specified with nullNested. The nullNested cannot be specified if the cutoff is specified.

revDirec
: Reverse the direction of deciding a power by fit indices (e.g., less than –> greater than). The default is to count the proportion of fit indices that indicates lower fit to the model, such as how many RMSEA in the alternative model that is worse than cutoffs. The direction can be reversed by setting as TRUE.

usedFit
: The vector of names of fit indices that researchers wish to get powers from. The default is to get powers of all fit indices

alpha
: The alpha level used to find the cutoff if the nullObject is specified. This argument is not applicable if the cutoff is specified.

nVal
: The sample size value that researchers wish to find the power from. This argument is applicable when altObject has a random sample size.

pmMCARval
: The percent missing completely at random value that researchers wish to find the power from. This argument is applicable when altObject has a random percent missing completely at random.

pmMARval
: The percent missing at random value that researchers wish to find the power from. This argument is applicable when altObject has a random percent missing at random.

condCutoff
: A logical value to use a conditional quantile method (if TRUE) or logistic regression method (if FALSE) to find the power. The conditional quantile method use quantile regression to find the quantile of the cutoff on the alternative sampling distribution that varies nVal, pmMCARval, or pmMARval. The value of 1 - quantile will be reported as the power given the set of nVal, pmMCARval, and pmMARval. The logistic regression method is based on transforming the fit indices value to reject/retain decision first. Then, the logistic regression is used to predict reject/retain decision given the set of nVal, pmMCARval, and pmMARval. The predicted probability is reported as a power. This argument is applicable for specification of cutoff only.

df
: The degree of freedom used in spline method in quantile regression (condCutoff = TRUE). If df is 0, which is the default, the spline method will not be applied.

Value

List of power given different fit indices. The TraditionalChi means the proportion of replications that are rejected by the traditional chi-square difference test.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- getCutoff to find the cutoffs from null model.
- SimResult to see how to create simResult

Examples

```
## Not run:
# Null model (Nested model) with one factor
loading.null <- matrix(0, 6, 1)
loading.null[1:6, 1] <- NA
LY.NULL <- bind(loading.null, 0.7)
RPS.NULL <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model.NULL <- model(LY = LY.NULL, RPS = RPS.NULL, RTE = RTE, modelType="CFA")

# Alternative model (Parent model) with two factors
loading.alt <- matrix(0, 6, 2)
loading.alt[1:3, 1] <- NA
loading.alt[4:6, 2] <- NA
LY.ALT <- bind(loading.alt, 0.7)
latent.cor.alt <- matrix(NA, 2, 2)
diag(latent.cor.alt) <- 1
RPS.ALT <- binds(latent.cor.alt, 0.7)
CFA.Model.ALT <- model(LY = LY.ALT, RPS = RPS.ALT, RTE = RTE, modelType="CFA")

# We make the examples running only 10 replications to save time.
# In reality, more replications are needed.
Output.NULL.NULL <- sim(10, n=500, model=CFA.Model.NULL, generate=CFA.Model.NULL)
Output.ALT.NULL <- sim(10, n=500, model=CFA.Model.NULL, generate=CFA.Model.ALT)
Output.NULL.ALT <- sim(10, n=500, model=CFA.Model.ALT, generate=CFA.Model.NULL)
Output.ALT.ALT <- sim(10, n=500, model=CFA.Model.ALT, generate=CFA.Model.ALT)

# Find the power based on the derived cutoff from the models analyzed on the null datasets
getPowerFitNested(Output.ALT.NULL, Output.ALT.ALT, nullNested=Output.NULL.NULL,
nullParent=Output.NULL.ALT)

# Find the power based on the chi-square value at df=1 and the CFI change (intentionally
# use a cutoff from Cheung and Rensvold (2002) in an appropriate situation).
getPowerFitNested(Output.ALT.NULL, Output.ALT.ALT, cutoff=c(Chi=3.84, CFI=-0.10))

# The example of continous varying sample size. Note that more fine-grained
# values of n is needed, e.g., n=seq(50, 500, 1)
Output.NULL.NULL2 <- sim(NULL, n=seq(50, 500, 50), model=CFA.Model.NULL, generate=CFA.Model.NULL)
Output.ALT.NULL2 <- sim(NULL, n=seq(50, 500, 50), model=CFA.Model.NULL, generate=CFA.Model.ALT)
Output.NULL.ALT2 <- sim(NULL, n=seq(50, 500, 50), model=CFA.Model.ALT, generate=CFA.Model.NULL)
Output.ALT.ALT2 <- sim(NULL, n=seq(50, 500, 50), model=CFA.Model.ALT, generate=CFA.Model.ALT)

# Get the power based on the derived cutoff from the null model at the sample size of 250
```

```
getPowerFitNested(Output.ALT.NULL2, Output.ALT.ALT2, nullNested=Output.NULL.NULL2,
nullParent=Output.NULL.ALT2, nVal = 250)

# Get the power based on the rule of thumb from the null model at the sample size of 250
getPowerFitNested(Output.ALT.NULL2, Output.ALT.ALT2, cutoff=c(Chi=3.84, CFI=-0.10), nVal = 250)

## End(Not run)
```

---

getPowerFitNonNested *Find power in rejecting non-nested models based on the differences in fit indices*

Description

Find the proportion of the difference in fit indices from one model that does not in the range of sampling distribution from another model (reject that the dataset comes from the second model) or indicates worse fit than a specified cutoff.

Usage

```
getPowerFitNonNested(dat2Mod1, dat2Mod2, cutoff = NULL, dat1Mod1 = NULL,
dat1Mod2 = NULL, revDirec = FALSE, usedFit = NULL, alpha = 0.05, nVal = NULL,
pmMCARval = NULL, pmMARval = NULL, condCutoff = TRUE, df = 0, onetailed = FALSE)
```

Arguments

`dat2Mod1`	SimResult that saves the simulation of analyzing Model 1 by datasets created from Model 2
`dat2Mod2`	SimResult that saves the simulation of analyzing Model 2 by datasets created from Model 2
`cutoff`	A vector of priori cutoffs for fit indices. The `cutoff` cannot be specified if the `dat1Mod1` and `dat1Mod2` are specified.
`dat1Mod1`	The SimResult that saves the simulation of analyzing Model 1 by datasets created from Model 1. This argument must be specified with `dat1Mod2`. The `dat1Mod1` cannot be specified if the `cutoff` is specified.
`dat1Mod2`	The SimResult that saves the simulation of analyzing Model 2 by datasets created from Model 1. This argument must be specified with `dat1Mod1`. The `dat1Mod2` cannot be specified if the `cutoff` is specified.
`revDirec`	Reverse the direction of deciding a power by fit indices (e.g., less than –> greater than). The default is to count the proportion of fit indices that indicates lower fit to the model, such as how many RMSEA in the alternative model that is worse than cutoffs. The direction can be reversed by setting as `TRUE`.
`usedFit`	The vector of names of fit indices that researchers wish to get powers from. The default is to get powers of all fit indices
`alpha`	The alpha level used to find the cutoff if the `nullObject` is specified. This argument is not applicable if the `cutoff` is specified.

nVal	The sample size value that researchers wish to find the power from. This argument is applicable when altObject has a random sample size.
pmMCARval	The percent missing completely at random value that researchers wish to find the power from. This argument is applicable when altObject has a random percent missing completely at random.
pmMARval	The percent missing at random value that researchers wish to find the power from. This argument is applicable when altObject has a random percent missing at random.
condCutoff	A logical value to use a conditional quantile method (if TRUE) or logistic regression method (if FALSE) to find the power. The conditional quantile method use quantile regression to find the quantile of the cutoff on the alternative sampling distribution that varies nVal, pmMCARval, or pmMARval. The value of 1 - quantile will be reported as the power given the set of nVal, pmMCARval, and pmMARval. The logistic regression method is based on transforming the fit indices value to reject/retain decision first. Then, the logistic regression is used to predict reject/retain decision given the set of nVal, pmMCARval, and pmMARval. The predicted probability is reported as a power. This argument is applicable for specification of cutoff only.
df	The degree of freedom used in spline method in quantile regression (condCutoff = TRUE). If df is 0, which is the default, the spline method will not be applied.
onetailed	Derive the cutoff by using one-tailed test if specified as TRUE. This argument is applicable only when dat1Mod1 and dat1Mod2 are specified.

Value

List of power given different fit indices.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- getCutoffNonNested to find the cutoffs for non-nested model comparison
- SimResult to see how to create simResult

Examples

```
## Not run:
# Model A: Factor 1 on Items 1-3 and Factor 2 on Items 4-8
loading.A <- matrix(0, 8, 2)
loading.A[1:3, 1] <- NA
loading.A[4:8, 2] <- NA
LY.A <- bind(loading.A, 0.7)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, "runif(1, 0.7, 0.9)")
RTE <- binds(diag(8))
```

```
CFA.Model.A <- model(LY = LY.A, RPS = RPS, RTE = RTE, modelType="CFA")

# Model B: Factor 1 on Items 1-4 and Factor 2 on Items 5-8
loading.B <- matrix(0, 8, 2)
loading.B[1:4, 1] <- NA
loading.B[5:8, 2] <- NA
LY.B <- bind(loading.B, 0.7)
CFA.Model.B <- model(LY = LY.B, RPS = RPS, RTE = RTE, modelType="CFA")

# The actual number of replications should be greater than 10.
Output.A.A <- sim(10, n=500, model=CFA.Model.A, generate=CFA.Model.A)
Output.A.B <- sim(10, n=500, model=CFA.Model.B, generate=CFA.Model.A)
Output.B.A <- sim(10, n=500, model=CFA.Model.A, generate=CFA.Model.B)
Output.B.B <- sim(10, n=500, model=CFA.Model.B, generate=CFA.Model.B)

# Find the power based on the derived cutoff for both models
getPowerFitNonNested(Output.B.A, Output.B.B, dat1Mod1=Output.A.A, dat1Mod2=Output.A.B)

# Find the power based on the AIC and BIC of 0 (select model B if Output.B.B has lower AIC or BIC)
getPowerFitNonNested(Output.B.A, Output.B.B, cutoff=c(AIC=0, BIC=0))

## End(Not run)
```

`imposeMissing` *Impose MAR, MCAR, planned missingness, or attrition on a data set*

Description

Function imposes missing values on a data based on the known missing data types, including MCAR, MAR, planned, and attrition.

Usage

```
impose(miss, data.mat, pmMCAR = NULL, pmMAR = NULL)
imposeMissing(data.mat, cov = 0, pmMCAR = 0, pmMAR = 0, nforms = 0,
itemGroups = list(), twoMethod = 0, prAttr = 0, timePoints = 1,
ignoreCols = 0, threshold = 0, logit = "", logical = NULL)
```

Arguments

`miss`	Missing data object (`SimMissing`) used as the template for impose missing values
`data.mat`	Data to impose missing upon. Can be either a matrix or a data frame.
`cov`	Column indices of a covariate to be used to impose MAR missing, or MAR attrition. Will not be included in any removal procedure. See details.
`pmMCAR`	Decimal percent of missingness to introduce completely at random on all variables.

pmMAR	Decimal percent of missingness to introduce using the listed covariate as predictor. See details.
nforms	The number of forms for planned missing data designs, not including the shared form.
itemGroups	List of lists of item groupings for planned missing data forms. Unless specified, items will be divided into groups sequentially (e.g. 1-3,4-6,7-9,10-12)
twoMethod	With missing on one variable: vector of (column index, percent missing). Will put a given percent missing on that column in the matrix to simulate a two method planned missing data research design. With missing on two or more variables: list of (column indices, percent missing).
prAttr	Probability (or vector of probabilities) of an entire case being removed due to attrition at a given time point. When a covariate is specified along with this argument, attrition will be predicted by the covariate (MAR attrition). See details.
timePoints	Number of timepoints items were measured over. For longitudinal data, planned missing designs will be implemented within each timepoint. All methods to impose missing values over time assume an equal number of variables at each time point.
ignoreCols	The columns not imposed any missing values for any missing data patterns.
threshold	The threshold of the covariate used to impose missing values. Values on the covariate above this threshold are eligible to be deleted. The default threshold is the mean of the variable.
logit	The script used for imposing missing values by logistic regression. The script is similar to the specification of regression in `lavaan` such that each line begins with a dependent variable, then '~' is used as regression sign, and the formula of a linear combination of independent variable plus constant, such as y1 ~ 0.5 + 0.2*y2. '#' and '!' can be used as a comment (like `lavaan`). For the intercept, users may use 'p()' to specify the average proportion of missing, such as y1 ~ p(0.2) + 0.3*y2, which the average missing proportion of y1 is 0.2 and the missing of y1 depends on y2. Users may visualize the missing proportion from the logistic specification by the `plotLogitMiss` function.
logical	A matrix of logical values (`TRUE/FALSE`). If a value in the dataset is corresponding to the `TRUE` in the logical matrix, the value will be missing.

Details

Without specifying any arguments, no missing values will be introduced.

A single covariate is required to specify MAR missing - this covariate can be distributed in any way. This covariate can be either continuous or categorical, as long as it is numerical. If the covariate is categorical, the threshold should be specified to one of the levels.

MAR missingness is specified using the threshold method - any value on the covariate that is above the specified threshold indicates a row eligible for deletion. If the specified total amount of MAR missingness is not possible given the total rows eligible based on the threshold, the function iteratively lowers the threshold until the total percent missing is possible.

Planned missingness is parameterized by the number of forms (n). This is used to divide the cases into n groups. If the column groupings are not specified, a naive method will be used that divides the columns into n+1 equal forms sequentially (1-4,5-9,10-13..), where the first group is the shared form.The first list of column indices given will be used as the shared group. If this is not desired, this list can be left empty.

For attrition, the probability can be specified as a single value or as a vector. For a single value, the probability of attrition will be the same across time points, and affects only cases not previously lost due to attrition. If this argument is a vector, this specifies different probabilities of attrition for each time point. Values will be recycled if this vector is smaller than the specified number of time points.

An MNAR processes can be generated by specifying MAR missingness and then dropping the covariate from the subsequent analysis.

Currently, if MAR missing is imposed along with attrition, both processes will use the same covariate and threshold.

Currently, all types of missingness (MCAR, MAR, planned, and attrition) are imposed independently. This means that specified global values of percent missing will not be additive (10 percent MCAR + 10 percent MAR does not equal 20 percent total missing).

Value

A data matrix with NAs introduced in the way specified by the arguments.

Author(s)

Patrick Miller (University of Kansas; <patr1ckm@ku.edu>), Alexander M. Schoemann (East Carolina University; <schoemanna@ecu.edu>)

See Also

- SimMissing for the alternative way to save missing data feature for using in the sim function.

Examples

```
data <- matrix(rep(rnorm(10,1,1),19),ncol=19)
datac <- cbind(data,rnorm(10,0,1),rnorm(10,5,5))

# Imposing Missing with the following arguments produces no missing values
imposeMissing(data)
imposeMissing(data,cov=c(1,2))
imposeMissing(data,pmMCAR=0)
imposeMissing(data,pmMAR=0)
imposeMissing(data,nforms=0)

#Some more usage examples

# No missing at variables 1 and 2
imposeMissing(data,cov=c(1,2),pmMCAR=.1)

# 3-Form design
imposeMissing(data,nforms=3)

# 3-Form design with specified groups of items (XABC)
```

```
imposeMissing(data, nforms = 3, itemGroups =
list(c(1,2,3,4,5), c(6,7,8,9,10), c(11,12,13,14,15), c(16,17,18,19)))

# 3-Form design when variables 20 and 21 are not missing
imposeMissing(datac,cov=c(20,21),nforms=3)

# 2 method design where the expensive measure is on Variable 19
imposeMissing(data,twoMethod=c(19,.8))

# Impose missing data with percent attrition of 0.1 in 5 time points
imposeMissing(datac,cov=21,prAttr=.1,timePoints=5)

# Logistic-regression MAR
colnames(data) <- paste("y", 1:ncol(data), sep="")
script <- 'y1 ~ 0.05 + 0.1*y2 + 0.3*y3
y4 ~ -2 + 0.1*y4
y5 ~ -0.5'
imposeMissing(data, logit=script)
```

`inspect` *Extract information from a simulation result*

Description

Extract information from a simulation result

Arguments

`object`	The target `SimResult` object
`what`	The target component to be extracted. Please see details below.
`improper`	Specify whether to include the information from the replications with improper solutions
`nonconverged`	Specify whether to include the information from the nonconvergent replications

Details

Here are the list of information that can be specified in the `what` argument. The items starting with * are the information that the `improper` and `nonconverged` arguments are not applicable.

- *`"modeltype"`: The type of the simulation result
- *`"nrep"`: The number of overall replications, including converged and nonconverged replications
- `"param"`: Parameter values (equivalent to the `getPopulation` function)
- `"stdparam"`: Standardized parameter values (equivalent to the `getPopulation` function with `std = TRUE`)
- `"coef"`: Parameter estimates (equivalent to the `coef` method)

- "se": Standard errors
- "fit": Fit indices
- "misspec": Misspecified parameter values
- "popfit": Population misfit
- "fmi1": Fraction missings type 1
- "fmi2": Fraction missings type 2
- "std": Standardized Parameter Estimates
- "stdse": Standard Errors of Standardized Values
- "cilower": Lower bounds of confidence intervals
- "ciupper": Upper bounds of confidence intervals
- "ciwidth": Widths of confidence intervals
- *"seed": Seed number (equivalent to the summarySeed function)
- "ngroup": Sample size of each group
- "ntotal": Total sample size
- "mcar": Percent missing completely at random
- "mar": Percent missing at random
- "extra": Extra output from the outfun argument from the sim function)
- *"time": Time elapsed in running the simulation (equivalent to the summaryTime function)
- *"converged": Convergence of each replication

Value

The target information depending on the what argument

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

SimResult for the object input

Examples

```
## Not run:
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
LY <- bind(loading, 0.7)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))
```

```
VY <- bind(rep(NA,6),2)

CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType = "CFA")

# In reality, more than 5 replications are needed.
Output <- sim(5, CFA.Model, n=200)
inspect(Output, "coef")
inspect(Output, "param")
inspect(Output, "se", improper = TRUE, nonconverged = TRUE)

## End(Not run)
```

---

likRatioFit — *Find the likelihood ratio (or Bayes factor) based on the bivariate distribution of fit indices*

Description

Find the log-likelihood of the observed fit indices on Model 1 and 2 from the real data on the bivariate sampling distribution of fit indices fitting Model 1 and Model 2 by the datasets from the Model 1 and Model 2. Then, the likelihood ratio is computed (which may be interpreted as posterior odd). If the prior odd is 1 (by default), the likelihood ratio is equivalent to Bayes Factor.

Usage

```
likRatioFit(outMod1, outMod2, dat1Mod1, dat1Mod2, dat2Mod1, dat2Mod2,
usedFit=NULL, prior=1)
```

Arguments

outMod1	`lavaan` that saves the analysis result of the first model from the target dataset
outMod2	`lavaan` that saves the analysis result of the second model from the target dataset
dat1Mod1	`SimResult` that saves the simulation of analyzing Model 1 by datasets created from Model 1
dat1Mod2	`SimResult` that saves the simulation of analyzing Model 2 by datasets created from Model 1
dat2Mod1	`SimResult` that saves the simulation of analyzing Model 1 by datasets created from Model 2
dat2Mod2	`SimResult` that saves the simulation of analyzing Model 2 by datasets created from Model 2
usedFit	Vector of names of fit indices that researchers wish to getCutoffs from. The default is to getCutoffs of all fit indices.
prior	The prior odds. The prior probability that Model 1 is correct over the prior probability that Model 2 is correct.

Value

The likelihood ratio (Bayes Factor) in preference of Model 1 to Model 2. If the value is greater than 1, Model 1 is preferred. If the value is less than 1, Model 2 is preferred.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

`SimResult` for a detail of simResult `pValueNested` for a nested model comparison by the difference in fit indices `pValueNonNested` for a nonnested model comparison by the difference in fit indices

Examples

```
## Not run:
# Model A; Factor 1 --> Factor 2; Factor 2 --> Factor 3
library(lavaan)
loading <- matrix(0, 11, 3)
loading[1:3, 1] <- NA
loading[4:7, 2] <- NA
loading[8:11, 3] <- NA
path.A <- matrix(0, 3, 3)
path.A[2, 1] <- NA
path.A[3, 2] <- NA
model.A <- estmodel(LY=loading, BE=path.A, modelType="SEM", indLab=c(paste("x", 1:3, sep=""),
paste("y", 1:8, sep="")))

out.A <- analyze(model.A, PoliticalDemocracy)

# Model A; Factor 1 --> Factor 3; Factor 3 --> Factor 2
path.B <- matrix(0, 3, 3)
path.B[3, 1] <- NA
path.B[2, 3] <- NA
model.B <- estmodel(LY=loading, BE=path.B, modelType="SEM", indLab=c(paste("x", 1:3, sep=""),
paste("y", 1:8, sep="")))

out.B <- analyze(model.B, PoliticalDemocracy)

loading.mis <- matrix("runif(1, -0.2, 0.2)", 11, 3)
loading.mis[is.na(loading)] <- 0

# Create SimSem object for data generation and data analysis template
datamodel.A <- model.lavaan(out.A, std=TRUE, LY=loading.mis)
datamodel.B <- model.lavaan(out.B, std=TRUE, LY=loading.mis)

# Get sample size
n <- nrow(PoliticalDemocracy)

# The actual number of replications should be greater than 20.
output.A.A <- sim(20, n=n, model.A, generate=datamodel.A)
output.A.B <- sim(20, n=n, model.B, generate=datamodel.A)
```

```
output.B.A <- sim(20, n=n, model.A, generate=datamodel.B)
output.B.B <- sim(20, n=n, model.B, generate=datamodel.B)

# Find the likelihood ratio ;The output may contain some warnings here.
# When the number of replications increases (e.g., 1000), the warnings should disappear.
likRatioFit(out.A, out.B, output.A.A, output.A.B, output.B.A, output.B.B)

## End(Not run)
```

---

miss *Specifying the missing template to impose on a dataset*

Description

Specifying the missing template (SimMissing) to impose on a dataset. The template will be used in Monte Carlo simulation such that, in the sim function, datasets are created and imposed by missing values created by this template. See imposeMissing for further details of each argument.

Usage

```
miss(cov = 0, pmMCAR = 0, pmMAR = 0, logit = "", nforms = 0, itemGroups = list(),
     timePoints = 1, twoMethod = 0, prAttr = 0, m = 0,
   package = "default", convergentCutoff = 0.8, ignoreCols = 0,
     threshold = 0, covAsAux = TRUE, logical = NULL, ...)
```

Arguments

cov	Column indices of any normally distributed covariates used in the data set.
pmMCAR	Decimal percent of missingness to introduce completely at random on all variables.
pmMAR	Decimal percent of missingness to introduce using the listed covariates as predictors.
logit	The script used for imposing missing values by logistic regression. The script is similar to the specification of regression in lavaan such that each line begins with a dependent variable, then '~' is used as regression sign, and the formula of a linear combination of independent variable plus constant, such as y1 ~ 0.5 + 0.2*y2. '#' and '!' can be used as a comment (like lavaan). For the intercept, users may use 'p()' to specify the average proportion of missing, such as y1 ~ p(0.2) + 0.3*y2, which the average missing proportion of y1 is 0.2 and the missing of y1 depends on y2. Users may visualize the missing proportion from the logistic specification by the plotLogitMiss function.
nforms	The number of forms for planned missing data designs, not including the shared form.
itemGroups	List of lists of item groupings for planned missing data forms. Without this, items will be divided into groups sequentially (e.g. 1-3,4-6,7-9,10-12)

timePoints
: Number of timepoints items were measured over. For longitudinal data, planned missing designs will be implemented within each timepoint.

twoMethod
: With missing on one variable: vector of (column index, percent missing). Will put a given percent missing on that column in the matrix to simulate a two method planned missing data research design. With missing on two or more variables: list of (column indices, percent missing).

prAttr
: Probability (or vector of probabilities) of an entire case being removed due to attrition at a given time point. See `imposeMissing` for further details.

m
: The number of imputations. The default is 0 such that the full information maximum likelihood is used.

package
: The package to be used in multiple imputation. The default value of this function is `"default"`. For the default option, if `m` is 0, the full information maximum likelihood is used. If `m` is greater than 0, the `"mice"` package is used. The possible inputs are `"default"`, `"Amelia"`, or `"mice"`.

convergentCutoff
: If the proportion of convergent results across imputations are greater than the specified value (the default is 80%), the analysis on the dataset is considered as convergent. Otherwise, the analysis is considered as nonconvergent. This attribute is applied for multiple imputation only.

ignoreCols
: The columns not imposed any missing values for any missing data patterns

threshold
: The threshold of covariates that divide between the area to impose missing and the area not to impose missing. The default threshold is the mean of the covariate.

covAsAux
: If `TRUE`, the covariate listed in the object will be used as auxiliary variables when putting in the model object. If `FALSE`, the covariate will be included in the analysis.

logical
: A matrix of logical values (`TRUE`/`FALSE`). If a value in the dataset is corresponding to the `TRUE` in the logical matrix, the value will be missing.

...
: Additional arguments used in multiple imputation function.

Value

A missing object that contains missing-data template (`SimMissing`)

Author(s)

Alexander M. Schoemann (East Carolina University; <schoemanna@ecu.edu>), Patrick Miller (University of Notre Dame; <pmille13@nd.edu>), Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `SimMissing` The resulting missing object

Examples

```
#Example of imposing 10% MCAR missing in all variables with no imputations (FIML method)
Missing <- miss(pmMCAR=0.1, ignoreCols="group")
summary(Missing)

loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, 0.7)
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

#Create data
dat <- generate(CFA.Model, n = 20)

#Impose missing
datmiss <- impose(Missing, dat)

#Analyze data
out <- analyze(CFA.Model, datmiss)
summary(out)

#Missing using logistic regression
script <- 'y1 ~ 0.05 + 0.1*y2 + 0.3*y3
y4 ~ -2 + 0.1*y4
y5 ~ -0.5'
Missing2 <- miss(logit=script, pmMCAR=0.1, ignoreCols="group")
summary(Missing2)
datmiss2 <- impose(Missing2, dat)

#Missing using logistic regression (2)
script <- 'y1 ~ 0.05 + 0.5*y3
y2 ~ p(0.2)
y3 ~ p(0.1) + -1*y1
y4 ~ p(0.3) + 0.2*y1 + -0.3*y2
y5 ~ -0.5'
Missing2 <- miss(logit=script)
summary(Missing2)
datmiss2 <- impose(Missing2, dat)

#Example to create simMissing object for 3 forms design at 3 timepoints with 10 imputations
Missing <- miss(nforms=3, timePoints=3, numImps=10)

#Missing template for data analysis with multiple imputation
Missing <- miss(package="mice", m=10, convergentCutoff=0.6)
```

`model` *Data generation template and analysis template for simulation.*

Description

This function creates a model template (lavaan parameter table), which can be used for data generation and/or analysis for simulated structural equation modeling using simsem. Models are specified using Y-side parameter matrices with LISREL syntax notation. Each parameter matrix must be a SimMatrix or SimVector built using bind. In addition to the usual Y-side matrices in LISREL, both PS and TE can be specified using correlation matrices (RPS, RTE) and scaled by a vector of residual variances (VTE, VPS) or total variances (VY, VE). Multiple group models can be created by passing lists of SimMatrix or SimVector to arguments, or by simply specifying the number of groups when all group models are identical.

Usage

```
model(LY = NULL, PS = NULL, RPS = NULL, TE = NULL, RTE = NULL, BE = NULL,
VTE = NULL, VY = NULL, VPS = NULL, VE = NULL, TY = NULL, AL = NULL, MY = NULL,
ME = NULL, KA = NULL, GA = NULL, modelType, indLab = NULL, facLab = NULL,
covLab = NULL, groupLab = "group", ngroups = 1, con = NULL)
model.cfa(LY = NULL,PS = NULL,RPS = NULL, TE = NULL,RTE = NULL, VTE = NULL,
VY = NULL, VPS = NULL, VE=NULL, TY = NULL, AL = NULL, MY = NULL, ME = NULL,
KA = NULL, GA = NULL, indLab = NULL, facLab = NULL, covLab = NULL,
groupLab = "group", ngroups = 1, con = NULL)
model.path(PS = NULL, RPS = NULL, BE = NULL, VPS = NULL, VE=NULL, AL = NULL,
ME = NULL, KA = NULL, GA = NULL, indLab = NULL, facLab = NULL, covLab = NULL,
groupLab = "group", ngroups = 1, con = NULL)
model.sem(LY = NULL,PS = NULL,RPS = NULL, TE = NULL,RTE = NULL, BE = NULL,
VTE = NULL, VY = NULL, VPS = NULL, VE=NULL, TY = NULL, AL = NULL, MY = NULL,
ME = NULL, KA = NULL, GA = NULL, indLab = NULL, facLab = NULL, covLab = NULL,
groupLab = "group", ngroups = 1, con = NULL)
```

Arguments

LY	Factor loading matrix from endogenous factors to Y indicators (must be SimMatrix object).
PS	Residual variance-covariance matrix among endogenous factors (must be SimMatrix object). Either RPS or PS (but not both) must be specified in SEM and CFA models.
RPS	Residual correlation matrix among endogenous factors (must be SimMatrix object). Either RPS or PS (but not both) must be specified in SEM and CFA models.
TE	Measurement error variance-covariance matrix among Y indicators (must be SimMatrix object). Either RTE or TE (but not both) must be specified in SEM and CFA models.
RTE	Measurement error correlation matrix among Y indicators (must be SimMatrix object). Either RTE or TE (but not both) must be specified in SEM and CFA models.
BE	Regression coefficient matrix among endogenous factors (must be SimMatrix object). BE must be specified in path analysis and SEM models.

VTE	Measurement error variance of indicators (must be SimVector object). Either VTE or VY (but not both) can be specified when RTE (instead of TE) is specified.
VY	Total variance of indicators (must be SimVector object). Either VTE or VY (but not both) can be specified when RTE (instead of TE) is specified.
VPS	Residual variance of factors (must be SimVector object). Either VPS or VE (but not both) can be specified when RPS (instead of PS) is specified.
VE	Total variance of of factors (must be SimVector object). Either VPS or VE (but not both) can be specified when RPS (instead of PS) is specified.
TY	Measurement intercepts of Y indicators (must be SimVector object). Either TY or MY (but not both) can be specified.
AL	Endogenous factor intercepts (must be SimVector object). Either AL or ME (but not both) can be specified.
MY	Y indicator means (must be SimVector object). Either TY or MY (but not both) can be specified.
ME	Total mean of endogenous factors (must be SimVector object). NOTE: Either endogenous factor intercept or total mean of endogenous factor is specified. Both cannot be simultaneously specified.
KA	Regression coefficient matrix from covariates to indicators (must be SimMatrix object). KA is needed when (fixed) exogenous covariates are needed only.
GA	Regression coefficient matrix from covariates to factors (must be SimMatrix object). GA is needed when (fixed) exogenous covariates are needed only.
modelType	"CFA", "Sem", or "Path". Model type must be specified to ensure that the matrices specified in model templates for data generation and analysis correspond to what the user intends.
indLab	Character vector of indicator labels. If left blank, automatic labels will be generated as `y1`, `y2`, ... `yy`.
facLab	Character vector of factor labels. If left blank, automatic labels will be generated as `f1`, `f2`, ... `ff`
covLab	Character vector of covariate labels. If left blank, automatic labels will be generated as `z1`, `z2`, ... `zz`
groupLab	Character of group-variable label (not the names of each group). If left blank, automatic labels will be generated as `group`
ngroups	Number of groups for data generation (defaults to 1). Should only be specified for multiple group models in which all parameter matrices are identical across groups (when ngroups > 1, specified matrices are replicated for all groups). For multiple group models in which parameter matrices differ among groups, parameter matrices should instead be specified as a list (if any matrix argument is a list, the number of groups will be equal to the list's length, and the ngroups argument will be ignored).
con	Additional parameters (phantom variables), equality constraints, and inequality constraints. Additional parameters must be specified using lavaan syntax. Allowed operators are ":=" (is defined as), "==" (is equal to), "<" (is less than),

and ">" (is greater than). Names used in syntax must correspond to labels defined on free parameters in the model (with the exception that the name to the left of ":=" is a new parameter name). On the right hand side of all operators, any mathematical expressions are allowed, e.g., `"newparam := (load1 + load2 + load3)/3"`. For the "<" and ">" operators in data generation, if the specified relation is at odds with parameter specifications (e.g., the parameter to the left of the ">" operator is less that the parameter to the right), the left hand side parameter will be changed so that the relation holds with a very small difference (i.e., 0.000001). For example, in "load1 > load2", if load1 is 0.5 and load2 is 0.6, load1 will be changed to 0.6 + 0.000001 = 0.600001.

Details

The *simsem* package is intricately tied to the *lavaan* package for analysis of structural equation models. The analysis template that is generated by `model` is a lavaan parameter table, a low-level access point to lavaan that allows repeated analyses to happen more rapidly. If desired, the parameter table generated can be used directly with lavaan for many analyses.

The data generation template is simply a list of `SimMatrix` or `SimVector` objects. The `SimSem` object can be passed to the function `generate` to generate data, or can be passed to the function `sim` to generate and/or analyze data.

To simulate multiple group data, users can either specify a integer in the ngroups argument (which creates a list of identical model arguments for each group), or pass a list of `SimMatrix` or `SimVector` to any of the matrix arguments with length(s) equal to the number of groups desired (this approach will cause the ngroups argument to be ignored). If only one argument is a list, all other arguments will be replicated across groups (with the same parameter identification, population parameter values/distributions, and misspecification). If equality constraints are specified, these parameters will be constrained to be equal across groups.

The `model.cfa`, `model.path`, and `model.sem` are the shortcuts for the `model` function when `modelType` are `"CFA"`, `"Path"`, and `"SEM"`, respectively.

Value

`SimSem` object that contains the data generation template (`@dgen`) and analysis template (`@pt`).

Author(s)

Patrick Miller (University of Notre Dame; <pmille13@nd.edu>), Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `sim` for simulations using the `SimSem` template.
- `generate` To generate data using the `SimSem` template.
- `analyze` To analyze real or generated data using the `SimSem` template.
- `draw` To draw parameters using the `SimSem` template.

Examples

```
# Example 1: Confirmatory factor analysis
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
LY <- bind(loading, 0.7)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))

VY <- bind(rep(NA,6),2)

CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType = "CFA")

# Example 2: Multiple-group CFA with weak invariance
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- paste0("con", 1:3)
loading[4:6, 2] <- paste0("con", 4:6)
LY <- bind(loading, 0.7)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))

VTE <- bind(rep(NA, 6), 0.51)

CFA.Model <- model(LY = LY, RPS = list(RPS, RPS), RTE = list(RTE, RTE), VTE=list(VTE, VTE),
ngroups=2, modelType = "CFA")

# Example 3: Linear growth curve model with model misspecification
factor.loading <- matrix(NA, 4, 2)
factor.loading[,1] <- 1
factor.loading[,2] <- 0:3
LY <- bind(factor.loading)

factor.mean <- rep(NA, 2)
factor.mean.starting <- c(5, 2)
AL <- bind(factor.mean, factor.mean.starting)

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

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

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

RTE <- binds(diag(4))

TY <- bind(rep(0, 4))

LCA.Model <- model(LY=LY, RPS=RPS, VPS=VPS, AL=AL, VTE=VTE, RTE=RTE, TY=TY, modelType="CFA")

# Example 4: Path analysis model with misspecified direct effect
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] <- "runif(1, 0.3, 0.5)"
starting.BE[4, 3] <- "runif(1,0.5,0.7)"
mis.path.BE <- matrix(0, 4, 4)
mis.path.BE[4, 1:2] <- "runif(1,-0.1,0.1)"
BE <- bind(path.BE, starting.BE, misspec=mis.path.BE)

residual.error <- diag(4)
residual.error[1,2] <- residual.error[2,1] <- NA
RPS <- binds(residual.error, "rnorm(1,0.3,0.1)")

ME <- bind(rep(NA, 4), 0)

Path.Model <- model(RPS = RPS, BE = BE, ME = ME, modelType="Path")

# Example 5: Full SEM model
loading <- matrix(0, 8, 3)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loading[7:8, 3] <- "con1"
loading.start <- matrix("", 8, 3)
loading.start[1:3, 1] <- 0.7
loading.start[4:6, 2] <- 0.7
loading.start[7:8, 3] <- "rnorm(1,0.6,0.05)"
LY <- bind(loading, loading.start)

RTE <- binds(diag(8))

factor.cor <- diag(3)
factor.cor[1, 2] <- factor.cor[2, 1] <- NA
RPS <- binds(factor.cor, 0.5)

path <- matrix(0, 3, 3)
path[3, 1:2] <- NA
path.start <- matrix(0, 3, 3)
path.start[3, 1] <- "rnorm(1,0.6,0.05)"
path.start[3, 2] <- "runif(1,0.3,0.5)"
BE <- bind(path, path.start)

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

```
# Shortcut example
SEM.model <- model.sem(BE=BE, LY=LY, RPS=RPS, RTE=RTE)

# Example 6: Multiple Group Model
loading1 <- matrix(NA, 6, 1)
LY1 <- bind(loading1, 0.7)
loading2 <- matrix(0, 6, 2)
loading2[1:3, 1] <- NA
loading2[4:6, 2] <- NA
LY2 <- bind(loading2, 0.7)

latent.cor2 <- matrix(NA, 2, 2)
diag(latent.cor2) <- 1
RPS1 <- binds(as.matrix(1))
RPS2 <- binds(latent.cor2, 0.5)

RTE <- binds(diag(6))

VTE <- bind(rep(NA, 6), 0.51)

noninvariance <- model(LY = list(LY1, LY2), RPS = list(RPS1, RPS2), RTE = list(RTE, RTE),
VTE=list(VTE, VTE), ngroups=2, modelType = "CFA")

# Example 7: Inequality Constraints

loading.in <- matrix(0, 6, 2)
loading.in[1:3, 1] <- c("load1", "load2", "load3")
loading.in[4:6, 2] <- c("load4", "load5", "load6")
mis <- matrix(0,6,2)
mis[loading.in == "0"] <- "runif(1, -0.1, 0.1)"
LY.in <- bind(loading.in, "runif(1, 0.7, 0.8)", mis)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))

VTE <- bind(rep(NA, 6), 0.51)

VPS1 <- bind(rep(1, 2))

VPS2 <- bind(rep(NA, 2), c(1.1, 1.2))

# Inequality constraint
script <- "
sth := load1 + load2 + load3
load4 == (load5 + load6) / 2
load4 > 0
load5 > 0
sth2 := load1 - load2
"
```

```
# Model Template
weak <- model(LY = LY.in, RPS = RPS, VPS=list(VPS1, VPS2), RTE = RTE, VTE=VTE, ngroups=2,
modelType = "CFA", con=script)
```

---

model.lavaan *Build the data generation template and analysis template from the lavaan result*

Description

Creates a data generation and analysis template (lavaan parameter table) for simulations with the `lavaan` result. Model misspecification may be added into the template by a vector, a matrix, or a list of vectors or matrices (for multiple groups).

Usage

```
model.lavaan(object, std = FALSE, LY = NULL, PS = NULL, RPS = NULL,
TE = NULL, RTE = NULL, BE = NULL, VTE = NULL, VY = NULL, VPS = NULL,
VE=NULL, TY = NULL, AL = NULL, MY = NULL, ME = NULL, KA = NULL,
GA = NULL)
```

Arguments

`object`	A `lavaan` object to be used to build the data generation and analysis template.
`std`	If TRUE, use the resulting standardized parameters for data generation. If FALSE, use the unstandardized parameters for data generation.
`LY`	Model misspecification in factor loading matrix from endogenous factors to Y indicators (need to be a matrix or a list of matrices).
`PS`	Model misspecification in residual covariance matrix among endogenous factors (need to be a symmetric matrix or a list of symmetric matrices).
`RPS`	Model misspecification in residual correlation matrix among endogenous factors (need to be a symmetric matrix or a list of symmetric matrices).
`TE`	Model misspecification in measurement error covariance matrix among Y indicators (need to be a symmetric matrix or a list of symmetric matrices).
`RTE`	Model misspecification in measurement error correlation matrix among Y indicators (need to be a symmetric matrix or a list of symmetric matrices).
`BE`	Model misspecification in regression coefficient matrix among endogenous factors (need to be a symmetric matrix or a list of symmetric matrices).
`VTE`	Model misspecification in measurement error variance of indicators (need to be a vector or a list of vectors).
`VY`	Model misspecification in total variance of indicators (need to be a vector or a list of vectors). NOTE: Either measurement error variance or indicator variance is specified. Both cannot be simultaneously specified.
`VPS`	Model misspecification in residual variance of factors (need to be a vector or a list of vectors).

VE
: Model misspecification in total variance of of factors (need to be a vector or a list of vectors). NOTE: Either residual variance of factors or total variance of factors is specified. Both cannot be simulatneously specified.

TY
: Model misspecification in measurement intercepts of Y indicators. (need to be a vector or a list of vectors).

AL
: Model misspecification in endogenous factor intercept (need to be a vector or a list of vectors).

MY
: Model misspecification in overall Y indicator means. (need to be a vector or a list of vectors). NOTE: Either measurement intercept of indicator mean can be specified. Both cannot be specified simultaneously.

ME
: Model misspecification in total mean of endogenous factors (need to be a vector or a list of vectors). NOTE: Either endogenous factor intercept or total mean of endogenous factor is specified. Both cannot be simultaneously specified.

KA
: Model misspecification in regression coefficient matrix from covariates to indicators (need to be a matrix or a list of matrices). KA is applicable when exogenous covariates are specified only.

GA
: Model misspecification in regression coefficient matrix from covariates to factors (need to be a matrix or a list of matrices). KA is applicable when exogenous covariates are specified only.

Value

SimSem object that contains the data generation template (@dgen) and analysis template (@pt).

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- model To build data generation and data analysis template for simulation.
- sim for simulations using the SimSem template.
- generate To generate data using the SimSem template.
- analyze To analyze real or generated data using the SimSem template.
- draw To draw parameters using the SimSem template.

Examples

```
library(lavaan)
HS.model <- ' visual  =~ x1 + x2 + x3
             textual =~ x4 + x5 + x6
             speed   =~ x7 + x8 + x9 '

fit <- cfa(HS.model, data=HolzingerSwineford1939)

# Create data generation and data analysis model from lavaan
# Data generation is based on standardized parameters
```

```
datamodel1 <- model.lavaan(fit, std=TRUE)

# Data generation is based on unstandardized parameters
datamodel2 <- model.lavaan(fit, std=FALSE)

# Data generation model with misspecification on cross-loadings
crossload <- matrix("runif(1, -0.1, 0.1)", 9, 3)
crossload[1:3, 1] <- 0
crossload[4:6, 2] <- 0
crossload[7:9, 3] <- 0
datamodel3 <- model.lavaan(fit, std=TRUE, LY=crossload)
```

`multipleAllEqual` *Test whether all objects are equal*

Description

Test whether all objects are equal. The test is based on the `all.equal` function.

Usage

```
multipleAllEqual(...)
```

Arguments

`...`	The target objects

Value

`TRUE` if all objects are equal.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

Examples

```
multipleAllEqual(1:5, 1:5, seq(2, 10, 2)/2) # Should be TRUE
multipleAllEqual(1:5, 1:6, seq(2, 10, 2)/2) # Should be FALSE
```

`plotCIwidth` *Plot a confidence interval width of a target parameter*

Description

Plot a confidence interval width of a target parameter

Usage

```
plotCIwidth(object, targetParam, assurance = 0.50, useContour = TRUE)
```

Arguments

`object`	The target (`SimResult`
`targetParam`	One or more target parameters to be plotted
`assurance`	The percentile of the resulting width. When assurance is 0.50, the median of the widths is provided. See Lai & Kelley (2011) for more details.
`useContour`	If there are two things from varying sample size, varying percent completely at random, or varying percent missing at random, the `plotCutoff` function will provide 3D graph. A contour graph is a default. However, if this is specified as `FALSE`, perspective plot is used.

Value

NONE. The plot the confidence interval width is provided.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

Lai, K., & Kelley, K. (2011). Accuracy in parameter estimation for targeted effects in structural equation modeling: Sample size planning for narrow confidence intervals. *Psychological Methods, 16*, 127-148.

See Also

- `SimResult` for simResult that used in this function.
- `getCIwidth` to get confidence interval widths

Examples

```
## Not run:
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loadingValues <- matrix(0, 6, 2)
loadingValues[1:3, 1] <- 0.7
loadingValues[4:6, 2] <- 0.7
LY <- bind(loading, loadingValues)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTE <- binds(error.cor)
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output <- sim(5, n=200, model=CFA.Model)

# Plot the widths of factor correlation
plotCIwidth(Output, "f1~~f2", assurance = 0.80)

# The example of continous varying sample size. Note that more fine-grained
# values of n is needed, e.g., n=seq(50, 500, 1)
Output2 <- sim(NULL, n=seq(450, 500, 10), model=CFA.Model)

# Plot the widths along sample size value
plotCIwidth(Output2, "f1~~f2", assurance = 0.80)

# Specify both continuous sample size and percent missing completely at random.
# Note that more fine-grained values of n and pmMCAR is needed, e.g., n=seq(50, 500, 1)
# and pmMCAR=seq(0, 0.2, 0.01)
Output3 <- sim(NULL, n=seq(450, 500, 10), pmMCAR=c(0, 0.05, 0.1, 0.15), model=CFA.Model)

# Plot the contours that each contour represents the value of widths at each level
# of sample size and percent missing completely at random
plotCIwidth(Output3, "f1~~f2", assurance = 0.80)

## End(Not run)
```

`plotCoverage` *Make a plot of confidence interval coverage rates*

Description

Make a plot of confidence interval coverage rates given varying parameters (e.g., sample size, percent missing completely at random, or random parameters in the model)

Usage

```
plotCoverage(object, coverParam, coverValue = NULL, contParam = NULL, contN = TRUE,
    contMCAR = TRUE, contMAR = TRUE, useContour = TRUE)
```

Arguments

object	SimResult that includes at least one randomly varying parameter (e.g. sample size, percent missing, model parameters)
coverParam	Vector of parameters names that the user wishes to find coverage rate for. This can be a vector of names (e.g., "f1=~y2", "f1~~f2").
coverValue	A target value used that users wish to find the coverage rate of that value (e.g., 0). If NULL, the parameter values will be used.
contParam	Vector of parameters names that vary over replications that users wish to use in the plot.
contN	Include the varying sample size in the coverage rate plot if available
contMCAR	Include the varying MCAR (missing completely at random percentage) in the coverage rate plot if available
contMAR	Include the varying MAR (missing at random percentage) in the coverage rate plot if available
useContour	This argument is used when users specify to plot two varying parameters. If TRUE, the contour plot is used. If FALSE, perspective plot is used.

Details

Predicting whether the confidence interval of each replication covers target value (or parameter) or not by varying parameters using logistic regression (without interaction). Then, plot the logistic curves predicting the probability of significance against the target varying parameters.

Value

Not return any value. This function will plot a graph only.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>), Alexander M. Schoemann (East Carolina University; <schoemanna@ecu.edu>)

See Also

- SimResult to see how to create a simResult object with randomly varying parameters.
- getCoverage to obtain a coverage rate given varying parameters values.

Examples

```
## Not run:
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, 0.4)
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# Specify both continuous sample size and percent missing completely at random.
# Note that more fine-grained values of n and pmMCAR is needed, e.g., n=seq(50, 500, 1)
# and pmMCAR=seq(0, 0.2, 0.01)

Output <- sim(NULL, n=seq(100, 200, 20), pmMCAR=c(0, 0.1, 0.2), model=CFA.Model)

# Plot the power of the first factor loading along the sample size value
plotCoverage(Output, "f1=~y1", contMCAR=FALSE)
plotCoverage(Output, "f1=~y1", coverValue = 0, contMCAR=FALSE)

# Plot the power of the correlation along the sample size and percent missing completely at random
plotCoverage(Output, "f1=~y1")

## End(Not run)
```

`plotCutoff` *Plot sampling distributions of fit indices with fit indices cutoffs*

Description

This function will plot sampling distributions of fit indices. The users may add cutoffs by specifying the `alpha` level.

Usage

```
plotCutoff(object, alpha = NULL, revDirec = FALSE, usedFit = NULL,
useContour = TRUE)
```

Arguments

`object`	The target (`SimResult`
`alpha`	A priori alpha level to get the cutoffs of fit indices
`revDirec`	The default is to find critical point on the side that indicates worse fit (the right side of RMSEA or the left side of CFI). If specifying as `TRUE`, the directions are reversed.
`usedFit`	The name of fit indices that researchers wish to plot
`useContour`	If there are two things from varying sample size, varying percent completely at random, or varying percent missing at random, the `plotCutoff` function will provide 3D graph. A contour graph is a default. However, if this is specified as `FALSE`, perspective plot is used.

Value

NONE. The plot the fit indices distributions is provided.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `SimResult` for simResult that used in this function.
- `getCutoff` to find values of cutoffs based on null hypothesis sampling distributions only

Examples

```
## Not run:
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loadingValues <- matrix(0, 6, 2)
loadingValues[1:3, 1] <- 0.7
loadingValues[4:6, 2] <- 0.7
LY <- bind(loading, loadingValues)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTE <- binds(error.cor)
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output <- sim(5, n=200, model=CFA.Model)

# Plot the cutoffs with desired fit indices
plotCutoff(Output, 0.05, usedFit=c("RMSEA", "SRMR", "CFI", "TLI"))

# The example of continous varying sample size. Note that more fine-grained
# values of n is needed, e.g., n=seq(50, 500, 1)
Output2 <- sim(NULL, n=seq(450, 500, 10), model=CFA.Model)

# Plot the cutoffs along sample size value
plotCutoff(Output2, 0.05)

# Specify both continuous sample size and percent missing completely at random.
# Note that more fine-grained values of n and pmMCAR is needed, e.g., n=seq(50, 500, 1)
# and pmMCAR=seq(0, 0.2, 0.01)
Output3 <- sim(NULL, n=seq(450, 500, 10), pmMCAR=c(0, 0.05, 0.1, 0.15), model=CFA.Model)

# Plot the contours that each contour represents the value of cutoff at each level
# of sample size and percent missing completely at random
```

```
plotCutoff(Output3, 0.05)

## End(Not run)
```

`plotCutoffNested` *Plot sampling distributions of the differences in fit indices between nested models with fit indices cutoffs*

Description

This function will plot sampling distributions of the differences in fit indices between nested models if the nested model is true. The users may add cutoffs by specifying the `alpha` level.

Usage

```
plotCutoffNested(nested, parent, alpha = 0.05, cutoff = NULL,
usedFit = NULL, useContour = T)
```

Arguments

`nested`	`SimResult` that saves the analysis results of nested model from multiple replications
`parent`	`SimResult` that saves the analysis results of parent model from multiple replications
`alpha`	A priori alpha level
`cutoff`	A priori cutoffs for fit indices, saved in a vector
`usedFit`	Vector of names of fit indices that researchers wish to plot the sampling distribution.
`useContour`	If there are two of sample size, percent completely at random, and percent missing at random are varying, the `plotCutoff` function will provide 3D graph. Contour graph is a default. However, if this is specified as `FALSE`, perspective plot is used.

Value

NONE. Only plot the fit indices distributions.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `SimResult` for simResult that used in this function.
- `getCutoffNested` to find the difference in fit indices cutoffs

Examples

```
## Not run:
# Nested model: One factor
loading.null <- matrix(0, 6, 1)
loading.null[1:6, 1] <- NA
LY.NULL <- bind(loading.null, 0.7)
RPS.NULL <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model.NULL <- model(LY = LY.NULL, RPS = RPS.NULL, RTE = RTE, modelType="CFA")

# Parent model: two factors
loading.alt <- matrix(0, 6, 2)
loading.alt[1:3, 1] <- NA
loading.alt[4:6, 2] <- NA
LY.ALT <- bind(loading.alt, 0.7)
latent.cor.alt <- matrix(NA, 2, 2)
diag(latent.cor.alt) <- 1
RPS.ALT <- binds(latent.cor.alt, "runif(1, 0.7, 0.9)")
CFA.Model.ALT <- model(LY = LY.ALT, RPS = RPS.ALT, RTE = RTE, modelType="CFA")

# The actual number of replications should be greater than 10.
Output.NULL.NULL <- sim(10, n=500, model=CFA.Model.NULL)
Output.NULL.ALT <- sim(10, n=500, model=CFA.Model.ALT, generate=CFA.Model.NULL)

# Plot the cutoffs in nested model comparison
plotCutoffNested(Output.NULL.NULL, Output.NULL.ALT, alpha=0.05)

## End(Not run)
```

`plotCutoffNonNested` *Plot sampling distributions of the differences in fit indices between non-nested models with fit indices cutoffs*

Description

This function will plot sampling distributions of the differences in fit indices between non-nested models. The users may add cutoffs by specifying the `alpha` level.

Usage

```
plotCutoffNonNested(dat1Mod1, dat1Mod2, dat2Mod1=NULL, dat2Mod2=NULL,
alpha=0.05, cutoff = NULL, usedFit = NULL, useContour = T, onetailed=FALSE)
```

Arguments

`dat1Mod1`	`SimResult` that saves the simulation of analyzing Model 1 by datasets created from Model 1
`dat1Mod2`	`SimResult` that saves the simulation of analyzing Model 2 by datasets created from Model 1

dat2Mod1	SimResult that saves the simulation of analyzing Model 1 by datasets created from Model 2
dat2Mod2	SimResult that saves the simulation of analyzing Model 2 by datasets created from Model 2
alpha	A priori alpha level
cutoff	A priori cutoffs for fit indices, saved in a vector
usedFit	Vector of names of fit indices that researchers wish to plot the sampling distribution.
useContour	If there are two of sample size, percent completely at random, and percent missing at random are varying, the plotCutoff function will provide 3D graph. Contour graph is a default. However, if this is specified as FALSE, perspective plot is used.
onetailed	If TRUE, the function will find the cutoff from one-tail test. If FALSE, the funciton will find the cutoff from two-tailed test.

Value

NONE. Only plot the fit indices distributions.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- SimResult for simResult that used in this function.
- getCutoffNonNested to find the difference in fit indices cutoffs for non-nested model comparison

Examples

```
## Not run:
# Model A: Factor 1 on Items 1-3 and Factor 2 on Items 4-8
loading.A <- matrix(0, 8, 2)
loading.A[1:3, 1] <- NA
loading.A[4:8, 2] <- NA
LY.A <- bind(loading.A, 0.7)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, "runif(1, 0.7, 0.9)")
RTE <- binds(diag(8))
CFA.Model.A <- model(LY = LY.A, RPS = RPS, RTE = RTE, modelType="CFA")

# Model B: Factor 1 on Items 1-4 and Factor 2 on Items 5-8
loading.B <- matrix(0, 8, 2)
loading.B[1:4, 1] <- NA
loading.B[5:8, 2] <- NA
LY.B <- bind(loading.B, 0.7)
CFA.Model.B <- model(LY = LY.B, RPS = RPS, RTE = RTE, modelType="CFA")
```

```
# The actual number of replications should be greater than 10.
Output.A.A <- sim(10, n=500, model=CFA.Model.A, generate=CFA.Model.A)
Output.A.B <- sim(10, n=500, model=CFA.Model.B, generate=CFA.Model.A)
Output.B.A <- sim(10, n=500, model=CFA.Model.A, generate=CFA.Model.B)
Output.B.B <- sim(10, n=500, model=CFA.Model.B, generate=CFA.Model.B)

# Plot cutoffs for both model A and model B
plotCutoffNonNested(Output.A.A, Output.A.B, Output.B.A, Output.B.B)

# Plot cutoffs for the model A only
plotCutoffNonNested(Output.A.A, Output.A.B)

# Plot cutoffs for the model A with one-tailed test
plotCutoffNonNested(Output.A.A, Output.A.B, onetailed=TRUE)

## End(Not run)
```

`plotDist` *Plot a distribution of a data distribution object*

Description

Plot a distribution of a data distribution object

Usage

```
plotDist(object, xlim = NULL, ylim = NULL, r = 0, var = NULL, contour = TRUE)
```

Arguments

`object`	The data distribution object (`SimDataDist`) to plot a distribution
`xlim`	A numeric vector with two elements specifying the lower and upper limit of the x-axis to be plotted.
`ylim`	A numeric vector with two elements specifying the lower and upper limit of the y-axis to be plotted. This argument is applicable for the joint distribution of two dimensions only
`r`	The correlation of two dimensions in the joint distribution
`var`	A vector of the index of variables to be plotted. The length of vector cannot be greater than 2.
`contour`	Applicable if two variables are used only. If TRUE, the contour plot is provided. If FALSE, the perspective plot is provided.

Value

No return value. This function will plot a graph only.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- SimDataDist for plotting a data distribution object

Examples

```
datadist <- bindDist(skewness = c(0, -2, 2), kurtosis = c(2, 4, 4))

# Plot the joint distribution of Variables 1 and 2 with correlation of 0.5
plotDist(datadist, r=0.5, var=1:2)

# Plot the marginal distribution of the variable 3
plotDist(datadist, var=3)

## Not run:
datadist2 <- bindDist(c("chisq", "t", "f"), list(df=5), list(df=3), list(df1=3, df2=5))

# Plot the joint distribution of Variables 1 and 2 with correlation of 0.5
plotDist(datadist2, r=0.5, var=1:2)

# Plot the marginal distribution of the variable 3
plotDist(datadist2, var=3)

## End(Not run)
```

---

plotLogitMiss — *Visualize the missing proportion when the logistic regression method is used.*

Description

Visualize the missing proportion when the logistic regression method is used. The maximum number of independent variables is 2. The extra independent variables will be fixed as a value (the default is 0).

Usage

```
plotLogitMiss(script, ylim=c(0,1), x1lim=c(-3,3), x2lim=c(-3,3), otherx=0,
useContour=TRUE)
```

Arguments

script — The script used in specifying missing data using the logistic regression. See further details in the `logit` argument of the `miss` function

ylim — The range of missing proportion to be plotted.

x1lim — The range of the first independent variable to be plotted

x2lim — The range of the second independent variable to be plotted

otherx — The value of the extra independent variables to be fixed as.

useContour — If there are two or more independent variables, the function will provide 3D graph. Contour graph is a default. However, if this is specified as `FALSE`, perspective plot is used.

Value

Not return any value. This function will plot a graph only. If the number of independent variable is 0, the bar graph is provided. If the number of independent variables is 1, the logistic curve is provided. If the number of independent variables is 2, contour or perspective plot is provided.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>), Alexander M. Schoemann (East Carolina University; <schoemanna@ecu.edu>)

See Also

- `miss` to create the missing data template
- `impose` to impose missing data

Examples

```
script <- 'y1 ~ 0.05 + 0.1*y2 + 0.3*y3
y4 ~ -2 + 0.1*y4
y5 ~ -0.5'
plotLogitMiss(script)

script2 <- 'y1 ~ 0.05 + 0.5*y3
y2 ~ p(0.2)
y3 ~ p(0.1) + -1*y1
y4 ~ p(0.3) + 0.2*y1 + -0.3*y2
y5 ~ -0.5'
plotLogitMiss(script2)
```

plotMisfit *Plot the population misfit in the result object*

Description

Plot a histogram of the amount of population misfit in parameter result object or the scatter plot of the relationship between misspecified parameter and the population misfit or the fit indices

Usage

```
plotMisfit(object, usedFit="default", misParam=NULL)
```

Arguments

object	The result object, SimResult
usedFit	The sample fit indices or population misfit used to plot. All sample fit indices are available. The available population misfit are "pop.f0", "pop.rmsea", and "pop.srmr". If the misParam is not specified, all population misfit are used. If the misParam is specified, the "pop.rmsea" is used in the plot.
misParam	The index or the name of misspecified parameters used to plot.

Value

None. This function will plot only.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

Examples

```
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] <- "runif(1, 0.3, 0.5)"
starting.BE[4, 3] <- "runif(1, 0.5, 0.7)"
mis.path.BE <- matrix(0, 4, 4)
mis.path.BE[4, 1:2] <- "runif(1, -0.1, 0.1)"
BE <- bind(path.BE, starting.BE, misspec=mis.path.BE)

residual.error <- diag(4)
residual.error[1,2] <- residual.error[2,1] <- NA
RPS <- binds(residual.error, "rnorm(1, 0.3, 0.1)")

ME <- bind(rep(NA, 4), 0)

Path.Model <- model(RPS = RPS, BE = BE, ME = ME, modelType="Path")
```

```
# The number of replications in actual analysis should be much more than 20
Output <- sim(20, n=500, Path.Model)

# Plot the distribution of population misfit
plotMisfit(Output)

# Plot the relationship between population RMSEA and all misspecified direct effects
plotMisfit(Output, misParam=1:2)

# Plot the relationship between sample CFI and all misspecified direct effects
plotMisfit(Output, usedFit="CFI", misParam=1:2)
```

`plotPower` *Make a power plot of a parameter given varying parameters*

Description

Make a power plot of a parameter given varying parameters (e.g., sample size, percent missing completely at random, or random parameters in the model)

Usage

```
plotPower(object, powerParam, alpha = 0.05, contParam = NULL, contN = TRUE,
contMCAR = TRUE, contMAR = TRUE, useContour=TRUE)
```

Arguments

`object`	`SimResult` that includes at least one randomly varying parameter (e.g. sample size, percent missing, model parameters)
`powerParam`	Vector of parameters names that the user wishes to find power for. This can be a vector of names (e.g., "f1=~y2", "f1~~f2").
`alpha`	Alpha level to use for power analysis.
`contParam`	Vector of parameters names that vary over replications that users wish to use in the plot.
`contN`	Include the varying sample size in the power plot if available
`contMCAR`	Include the varying MCAR (missing completely at random percentage) in the power plot if available
`contMAR`	Include the varying MAR (missing at random percentage) in the power plot if available
`useContour`	This argument is used when users specify to plot two varying parameters. If `TRUE`, the contour plot is used. If `FALSE`, perspective plot is used.

Details

Predicting whether each replication is significant or not by varying parameters using logistic regression (without interaction). Then, plot the logistic curves predicting the probability of significance against the target varying parameters.

Value

Not return any value. This function will plot a graph only.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>), Alexander M. Schoemann (East Carolina University; <schoemanna@ecu.edu>)

See Also

- `SimResult` to see how to create a simResult object with randomly varying parameters.
- `getPower` to obtain a statistical power given varying parameters values.

Examples

```
## Not run:
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, 0.4)
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# Specify both continuous sample size and percent missing completely at random.
# Note that more fine-grained values of n and pmMCAR is needed, e.g., n=seq(50, 500, 1)
# and pmMCAR=seq(0, 0.2, 0.01)

Output <- sim(NULL, n=seq(100, 200, 20), pmMCAR=c(0, 0.1, 0.2), model=CFA.Model)

# Plot the power of the first factor loading along the sample size value
plotPower(Output, "f1=~y1", contMCAR=FALSE)

# Plot the power of the correlation along the sample size and percent missing completely at random
plotPower(Output, "f1=~y1")

## End(Not run)
```

---

plotPowerFit — *Plot sampling distributions of fit indices that visualize power of rejecting datasets underlying misspecified models*

Description

This function will plot sampling distributions of fit indices that visualize power in rejecting the misspecified models

Usage

```
plotPowerFit(altObject, nullObject = NULL, cutoff = NULL, usedFit = NULL,
alpha = 0.05, contN = TRUE, contMCAR = TRUE, contMAR = TRUE,
useContour = TRUE, logistic = TRUE)
```

Arguments

`altObject`	The result object (`SimResult`) saves the simulation result of fitting the hypothesized model when the hypothesized model is `FALSE`.
`nullObject`	The result object (`SimResult`) saves the simulation result of fitting the hypothesized model when the hypothesized model is `TRUE`. This argument may be not specified if the `cutoff` is specified.
`cutoff`	A vector of priori cutoffs for fit indices.
`usedFit`	Vector of names of fit indices that researchers wish to plot.
`alpha`	A priori alpha level
`contN`	Include the varying sample size in the power plot if available
`contMCAR`	Include the varying MCAR (missing completely at random percentage) in the power plot if available
`contMAR`	Include the varying MAR (missing at random percentage) in the power plot if available
`useContour`	If there are two of sample size, percent completely at random, and percent missing at random are varying, the `plotCutoff` function will provide 3D graph. Contour graph is a default. However, if this is specified as `FALSE`, perspective plot is used.
`logistic`	If `logistic` is `TRUE` and the varying parameter exists (e.g., sample size or percent missing), the plot based on logistic regression predicting the significance by the varying parameters is preferred. If `FALSE`, the overlaying scatterplot with a line of cutoff is plotted.

Value

NONE. Only plot the fit indices distributions.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `SimResult` for simResult that used in this function.
- `getCutoff` to find values of cutoffs based on null hypothesis sampling distributions only
- `getPowerFit` to find power of rejecting the hypothesized model when the hypothesized model is `FALSE`.

Examples

```
## Not run:
# Null model: One factor model
loading.null <- matrix(0, 6, 1)
loading.null[1:6, 1] <- NA
LY.NULL <- bind(loading.null, 0.7)
RPS.NULL <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model.NULL <- model(LY = LY.NULL, RPS = RPS.NULL, RTE = RTE, modelType="CFA")

# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output.NULL <- sim(50, n=50, model=CFA.Model.NULL, generate=CFA.Model.NULL)

# Alternative model: Two-factor model
loading.alt <- matrix(0, 6, 2)
loading.alt[1:3, 1] <- NA
loading.alt[4:6, 2] <- NA
LY.ALT <- bind(loading.alt, 0.7)
latent.cor.alt <- matrix(NA, 2, 2)
diag(latent.cor.alt) <- 1
RPS.ALT <- binds(latent.cor.alt, 0.5)
CFA.Model.ALT <- model(LY = LY.ALT, RPS = RPS.ALT, RTE = RTE, modelType="CFA")
Output.ALT <- sim(50, n=50, model=CFA.Model.NULL, generate=CFA.Model.ALT)

# Plot the power based on derived cutoff from the null model using four fit indices
plotPowerFit(Output.ALT, nullObject=Output.NULL, alpha=0.05,
usedFit=c("RMSEA", "CFI", "TLI", "SRMR"))

# Plot the power of rejecting null model when the rule of thumb from Hu & Bentler (1999) is used
Rule.of.thumb <- c(RMSEA=0.05, CFI=0.95, TLI=0.95, SRMR=0.06)
plotPowerFit(Output.ALT, cutoff=Rule.of.thumb, alpha=0.05,
usedFit=c("RMSEA", "CFI", "TLI", "SRMR"))

# The example of continous varying sample size. Note that more fine-grained
# values of n is needed, e.g., n=seq(50, 500, 1)
Output.NULL2 <- sim(NULL, n=seq(50, 250, 25), model=CFA.Model.NULL, generate=CFA.Model.NULL)
Output.ALT2 <- sim(NULL, n=seq(50, 250, 25), model=CFA.Model.NULL, generate=CFA.Model.ALT)

# Plot the power based on derived cutoff from the null model using four fit indices
# along sample size
plotPowerFit(Output.ALT2, nullObject=Output.NULL2, alpha=0.05,
usedFit=c("RMSEA", "CFI", "TLI", "SRMR"))

# Plot the power based on rule of thumb along sample size
plotPowerFit(Output.ALT2, cutoff=Rule.of.thumb, alpha=0.05,
usedFit=c("RMSEA", "CFI", "TLI", "SRMR"))

## End(Not run)
```

`plotPowerFitNested` *Plot power of rejecting a nested model in a nested model comparison by each fit index*

Description

This function will plot sampling distributions of the differences in fit indices between parent and nested models. Two sampling distributions will be compared: nested model is `FALSE` (alternative model) and nested model is `TRUE` (null model).

Usage

```
plotPowerFitNested(altNested, altParent, nullNested = NULL,
nullParent = NULL, cutoff = NULL, usedFit = NULL, alpha = 0.05,
contN = TRUE, contMCAR = TRUE, contMAR = TRUE, useContour = TRUE,
logistic = TRUE)
```

Arguments

`altNested`	`SimResult` that saves the simulation result of the nested model when the nested model is `FALSE`.
`altParent`	`SimResult` that saves the simulation result of the parent model when the nested model is `FALSE`.
`nullNested`	`SimResult` that saves the simulation result of the nested model when the nested model is `TRUE`. This argument may not be specified if the `cutoff` is specified.
`nullParent`	`SimResult` that saves the simulation result of the parent model when the nested model is `TRUE`. This argument may not be specified if the `cutoff` is specified.
`cutoff`	A vector of priori cutoffs for the differences in fit indices.
`usedFit`	Vector of names of fit indices that researchers wish to plot.
`alpha`	A priori alpha level
`contN`	Include the varying sample size in the power plot if available
`contMCAR`	Include the varying MCAR (missing completely at random percentage) in the power plot if available
`contMAR`	Include the varying MAR (missing at random percentage) in the power plot if available
`useContour`	If there are two of sample size, percent completely at random, and percent missing at random are varying, the `plotCutoff` function will provide 3D graph. Contour graph is a default. However, if this is specified as `FALSE`, perspective plot is used.
`logistic`	If `logistic` is `TRUE` and the varying parameter exists (e.g., sample size or percent missing), the plot based on logistic regression predicting the significance by the varying parameters is preferred. If `FALSE`, the overlaying scatterplot with a line of cutoff is plotted.

Value

NONE. Only plot the fit indices distributions.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `SimResult` for simResult that used in this function.
- `getCutoffNested` to find the cutoffs of the differences in fit indices
- `plotCutoffNested` to visualize the cutoffs of the differences in fit indices
- `getPowerFitNested` to find the power in rejecting the nested model by the difference in fit indices cutoffs

Examples

```
## Not run:
# Null model: One-factor model
loading.null <- matrix(0, 6, 1)
loading.null[1:6, 1] <- NA
LY.NULL <- bind(loading.null, 0.7)
RPS.NULL <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model.NULL <- model(LY = LY.NULL, RPS = RPS.NULL, RTE = RTE, modelType="CFA")

# Alternative model: Two-factor model
loading.alt <- matrix(0, 6, 2)
loading.alt[1:3, 1] <- NA
loading.alt[4:6, 2] <- NA
LY.ALT <- bind(loading.alt, 0.7)
latent.cor.alt <- matrix(NA, 2, 2)
diag(latent.cor.alt) <- 1
RPS.ALT <- binds(latent.cor.alt, 0.7)
CFA.Model.ALT <- model(LY = LY.ALT, RPS = RPS.ALT, RTE = RTE, modelType="CFA")

# In reality, more than 10 replications are needed
Output.NULL.NULL <- sim(10, n=500, model=CFA.Model.NULL, generate=CFA.Model.NULL)
Output.ALT.NULL <- sim(10, n=500, model=CFA.Model.NULL, generate=CFA.Model.ALT)
Output.NULL.ALT <- sim(10, n=500, model=CFA.Model.ALT, generate=CFA.Model.NULL)
Output.ALT.ALT <- sim(10, n=500, model=CFA.Model.ALT, generate=CFA.Model.ALT)

# Plot the power based on the derived cutoff from the models analyzed on the null datasets
plotPowerFitNested(Output.ALT.NULL, Output.ALT.ALT, nullNested=Output.NULL.NULL,
nullParent=Output.NULL.ALT)

# Plot the power by only CFI
plotPowerFitNested(Output.ALT.NULL, Output.ALT.ALT, nullNested=Output.NULL.NULL,
nullParent=Output.NULL.ALT, usedFit="CFI")
```

```
# The example of continous varying sample size. Note that more fine-grained
# values of n is needed, e.g., n=seq(50, 500, 1)
Output.NULL.NULL2 <- sim(NULL, n=seq(50, 500, 5), model=CFA.Model.NULL, generate=CFA.Model.NULL)
Output.ALT.NULL2 <- sim(NULL, n=seq(50, 500, 5), model=CFA.Model.NULL, generate=CFA.Model.ALT)
Output.NULL.ALT2 <- sim(NULL, n=seq(50, 500, 5), model=CFA.Model.ALT, generate=CFA.Model.NULL)
Output.ALT.ALT2 <- sim(NULL, n=seq(50, 500, 5), model=CFA.Model.ALT, generate=CFA.Model.ALT)

# Plot logistic line for the power based on the derived cutoff from the null model
# along sample size values
plotPowerFitNested(Output.ALT.NULL2, Output.ALT.ALT2, nullNested=Output.NULL.NULL2,
nullParent=Output.NULL.ALT2)

# Plot scatterplot for the power based on the derived cutoff from the null model
# along sample size values
plotPowerFitNested(Output.ALT.NULL2, Output.ALT.ALT2, nullNested=Output.NULL.NULL2,
nullParent=Output.NULL.ALT2, logistic=FALSE)

# Plot scatterplot for the power based on the advanced CFI value
plotPowerFitNested(Output.ALT.NULL2, Output.ALT.ALT2, cutoff=c(CFI=-0.1), logistic=FALSE)

## End(Not run)
```

`plotPowerFitNonNested` *Plot power of rejecting a non-nested model based on a difference in fit index*

Description

Plot the proportion of the difference in fit indices from one model that does not in the range of sampling distribution from another model (reject that the dataset comes from the second model) or indicates worse fit than a specified cutoff. This plot can show the proportion in the second model that does not in the range of sampling distribution from the first model too.

Usage

```
plotPowerFitNonNested(dat2Mod1, dat2Mod2, dat1Mod1=NULL, dat1Mod2=NULL,
cutoff = NULL, usedFit = NULL, alpha = 0.05, contN = TRUE, contMCAR = TRUE,
contMAR = TRUE, useContour = TRUE, logistic = TRUE, onetailed = FALSE)
```

Arguments

`dat2Mod1`	`SimResult` that saves the simulation of analyzing Model 1 by datasets created from Model 2
`dat2Mod2`	`SimResult` that saves the simulation of analyzing Model 2 by datasets created from Model 2
`dat1Mod1`	`SimResult` that saves the simulation of analyzing Model 1 by datasets created from Model 1

dat1Mod2	SimResult that saves the simulation of analyzing Model 2 by datasets created from Model 1
cutoff	A vector of priori cutoffs for the differences in fit indices.
usedFit	Vector of names of fit indices that researchers wish to plot.
alpha	A priori alpha level
contN	Include the varying sample size in the power plot if available
contMCAR	Include the varying MCAR (missing completely at random percentage) in the power plot if available
contMAR	Include the varying MAR (missing at random percentage) in the power plot if available
useContour	If there are two of sample size, percent completely at random, and percent missing at random are varying, the `plotCutoff` function will provide 3D graph. Contour graph is a default. However, if this is specified as `FALSE`, perspective plot is used.
logistic	If `logistic` is `TRUE` and the varying parameter exists (e.g., sample size or percent missing), the plot based on logistic regression predicting the significance by the varying parameters is preferred. If `FALSE`, the overlaying scatterplot with a line of cutoff is plotted.
onetailed	If `TRUE`, the function will use the cutoff from one-tail test. If `FALSE`, the funciton will use the cutoff from two-tailed test.

Value

NONE. Only plot the fit indices distributions.

Author(s)

Sunthud Pornprasertmanit (`<psunthud@gmail.com>`)

See Also

- `SimResult` for simResult that used in this function.
- `getCutoffNonNested` to find the cutoffs of the differences in fit indices for non-nested model comparison
- `plotCutoffNonNested` to visualize the cutoffs of the differences in fit indices for non-nested model comparison
- `getPowerFitNonNested` to find the power in rejecting the non-nested model by the difference in fit indices cutoffs

Examples

```
## Not run:
# Model A: Factor 1 on Items 1-3 and Factor 2 on Items 4-8
loading.A <- matrix(0, 8, 2)
loading.A[1:3, 1] <- NA
loading.A[4:8, 2] <- NA
```

```
LY.A <- bind(loading.A, 0.7)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, "runif(1, 0.7, 0.9)")
RTE <- binds(diag(8))
CFA.Model.A <- model(LY = LY.A, RPS = RPS, RTE = RTE, modelType="CFA")

# Model B: Factor 1 on Items 1-4 and Factor 2 on Items 5-8
loading.B <- matrix(0, 8, 2)
loading.B[1:4, 1] <- NA
loading.B[5:8, 2] <- NA
LY.B <- bind(loading.B, 0.7)
CFA.Model.B <- model(LY = LY.B, RPS = RPS, RTE = RTE, modelType="CFA")

# The actual number of replications should be greater than 10.
Output.A.A <- sim(10, n=500, model=CFA.Model.A, generate=CFA.Model.A)
Output.A.B <- sim(10, n=500, model=CFA.Model.B, generate=CFA.Model.A)
Output.B.A <- sim(10, n=500, model=CFA.Model.A, generate=CFA.Model.B)
Output.B.B <- sim(10, n=500, model=CFA.Model.B, generate=CFA.Model.B)

# Plot the power based on the derived cutoff for both models
plotPowerFitNonNested(Output.B.A, Output.B.B, dat1Mod1=Output.A.A, dat1Mod2=Output.A.B)

# Plot the power based on AIC and BIC cutoffs
plotPowerFitNonNested(Output.B.A, Output.B.B, cutoff=c(AIC=0, BIC=0))

## End(Not run)
```

---

popDiscrepancy *Find the discrepancy value between two means and covariance matrices*

Description

Find the discrepancy value between two means and covariance matrices. See the definition of each index at summaryMisspec.

Usage

```
popDiscrepancy(paramM, paramCM, misspecM, misspecCM)
```

Arguments

paramM	The model-implied mean from the real parameters
paramCM	The model-implied covariance matrix from the real parameters
misspecM	The model-implied mean from the real and misspecified parameters
misspecCM	The model-implied covariance matrix from the real and misspecified parameters

Value

The discrepancy between two means and covariance matrices

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

Browne, M. W., & Cudeck, R. (1992). Alternative ways of assessing model fit. *Sociological Methods & Research, 21*, 230-258.

Examples

```
m1 <- rep(0, 3)
m2 <- c(0.1, -0.1, 0.05)
S1 <- matrix(c(1, 0.6, 0.5, 0.6, 1, 0.4, 0.5, 0.4, 1), 3, 3)
S2 <- matrix(c(1, 0.55, 0.55, 0.55, 1, 0.55, 0.55, 0.55, 1), 3, 3)
popDiscrepancy(m1, S1, m2, S2)
```

---

popMisfitMACS — *Find population misfit by sufficient statistics*

Description

Find the value quantifying the amount of population misfit: F_0, RMSEA, and SRMR. See the definition of each index at summaryMisspec.

Usage

```
popMisfitMACS(paramM, paramCM, misspecM, misspecCM, dfParam=NULL, fit.measures="all")
```

Arguments

paramM	The model-implied mean from the real parameters
paramCM	The model-implied covariance matrix from the real parameters
misspecM	The model-implied mean from the real and misspecified parameters
misspecCM	The model-implied covariance matrix from the real and misspecified parameters
dfParam	The degree of freedom of the real model
fit.measures	The names of indices used to calculate population misfit. There are three types of misfit: 1) discrepancy function ("f0"; see popDiscrepancy), 2) root mean squared error of approximation ("rmsea"; Equation 12 in Browne & Cudeck, 1992), and 3) standardized root mean squared residual ("srmr")

Value

The vector of the misfit indices

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

Browne, M. W., & Cudeck, R. (1992). Alternative ways of assessing model fit. *Sociological Methods & Research, 21*, 230-258.

Examples

```
m1 <- rep(0, 3)
m2 <- c(0.1, -0.1, 0.05)
S1 <- matrix(c(1, 0.6, 0.5, 0.6, 1, 0.4, 0.5, 0.4, 1), 3, 3)
S2 <- matrix(c(1, 0.55, 0.55, 0.55, 1, 0.55, 0.55, 0.55, 1), 3, 3)
popMisfitMACS(m1, S1, m2, S2)
```

---

pValue — *Find p-values (1 - percentile) by comparing a single analysis output from the result object*

Description

This function will provide *p* value from comparing a lavaan) or a OpenMx result from the simulation result (in SimResult).

Usage

```
pValue(target, dist, usedFit = NULL, nVal = NULL, pmMCARval = NULL,
pmMARval = NULL, df = 0)
```

Arguments

target	A value, multiple values, a lavaan object, or an OpenMx object used to find p values. This argument could be a cutoff of a fit index.
dist	The comparison distribution, which can be a vector of numbers, a data frame, or a result object.
usedFit	The vector of names of fit indices that researchers wish to find the *p* value from.
nVal	The sample size value that researchers wish to find the fit indices cutoffs from
pmMCARval	The percent missing completely at random value that researchers wish to find the fit indices cutoffs from.
pmMARval	The percent missing at random value that researchers wish to find the fit indices cutoffs from.
df	The degree of freedom used in spline method in predicting the fit indices by the predictors. If `df` is 0, the spline method will not be applied.

Details

In comparing fit indices, the *p* value is the proportion of the number of replications that provide poorer fit (e.g., less CFI value or greater RMSEA value) than the analysis result from the observed data.

Value

The *p* values of fit indices are provided, as well as two additional 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.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `SimResult` to run a simulation study

Examples

```
## Not run:
# Compare an analysis result with a result of simulation study
library(lavaan)
loading <- matrix(0, 9, 3)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loading[7:9, 3] <- NA
targetmodel <- estmodel(LY=loading, modelType="CFA", indLab=paste("x", 1:9, sep=""))
out <- analyze(targetmodel, HolzingerSwineford1939)

loading.trivial <- matrix("runif(1, -0.2, 0.2)", 9, 3)
loading.trivial[is.na(loading)] <- 0
mismodel <- model.lavaan(out, std=TRUE, LY=loading.trivial)

# The actual number of replications should be much greater than 20.
simout <- sim(20, n=nrow(HolzingerSwineford1939), mismodel)

# Find the p-value comparing the observed fit indices against the simulated
# sampling distribution of fit indices
pValue(out, simout)

## End(Not run)
```

pValueNested — *Find p-values (1 - percentile) for a nested model comparison*

Description

This function will provide *p* value from comparing the differences in fit indices between nested models with the simulation results of both parent and nested models when the nested model is true.

Usage

```
pValueNested(outNested, outParent, simNested, simParent, usedFit = NULL,
nVal = NULL, pmMCARval = NULL, pmMARval = NULL, df = 0)
```

Arguments

`outNested`	`lavaan` that saves the analysis result of the nested model from the target dataset
`outParent`	`lavaan` that saves the analysis result of the parent model from the target dataset
`simNested`	`SimResult` that saves the analysis results of nested model from multiple replications
`simParent`	`SimResult` that saves the analysis results of parent model from multiple replications
`usedFit`	Vector of names of fit indices that researchers wish to getCutoffs from. The default is to getCutoffs of all fit indices.
`nVal`	The sample size value that researchers wish to find the *p* value from.
`pmMCARval`	The percent missing completely at random value that researchers wish to find the *p* value from.
`pmMARval`	The percent missing at random value that researchers wish to find the the *p* value from.
`df`	The degree of freedom used in spline method in predicting the fit indices by the predictors. If `df` is 0, the spline method will not be applied.

Details

In comparing fit indices, the *p* value is the proportion of the number of replications that provide less preference for nested model (e.g., larger negative difference in CFI values or larger positive difference in RMSEA values) than the analysis result from the observed data.

Value

This function provides a vector of *p* values based on the comparison of the difference in fit indices from the real data with the simulation result. The *p* values of fit indices are provided, as well as two additional 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.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `SimResult` to run a simulation study

Examples

```
## Not run:
library(lavaan)

# Nested Model: Linear growth curve model
LY <- matrix(1, 4, 2)
LY[,2] <- 0:3
PS <- matrix(NA, 2, 2)
TY <- rep(0, 4)
AL <- rep(NA, 2)
TE <- diag(NA, 4)
nested <- estmodel(LY=LY, PS=PS, TY=TY, AL=AL, TE=TE, modelType="CFA",
indLab=paste("t", 1:4, sep=""))

# Parent Model: Unconditional growth curve model
LY2 <- matrix(1, 4, 2)
LY2[,2] <- c(0, NA, NA, 3)
parent <- estmodel(LY=LY2, PS=PS, TY=TY, AL=AL, TE=TE, modelType="CFA",
indLab=paste("t", 1:4, sep=""))

# Analyze the output
outNested <- analyze(nested, Demo.growth)
outParent <- analyze(parent, Demo.growth)

# Create data template from the nested model with small misfit on the linear curve
loadingMis <- matrix(0, 4, 2)
loadingMis[2:3, 2] <- "runif(1, -0.1, 0.1)"
datamodel <- model.lavaan(outNested, LY=loadingMis)

# Get the sample size
n <- nrow(Demo.growth)

# The actual replications should be much greater than 30.
simNestedNested <- sim(30, n=n, nested, generate=datamodel)
simNestedParent <- sim(30, n=n, parent, generate=datamodel)

# Find the p-value comparing the observed fit indices against the simulated
# sampling distribution of fit indices
```

```
pValueNested(outNested, outParent, simNestedNested, simNestedParent)

## End(Not run)
```

---

pValueNonNested *Find p-values (1 - percentile) for a non-nested model comparison*

Description

This function will provide *p* value from comparing the results of fitting real data into two models against the simulation from fitting the simulated data from both models into both models. The *p* values from both sampling distribution under the datasets from the first and the second models are reported.

Usage

```
pValueNonNested(outMod1, outMod2, dat1Mod1, dat1Mod2, dat2Mod1, dat2Mod2,
usedFit = NULL, nVal = NULL, pmMCARval = NULL, pmMARval = NULL, df = 0,
onetailed=FALSE)
```

Arguments

outMod1	lavaan that saves the analysis result of the first model from the target dataset
outMod2	lavaan that saves the analysis result of the second model from the target dataset
dat1Mod1	SimResult that saves the simulation of analyzing Model 1 by datasets created from Model 1
dat1Mod2	SimResult that saves the simulation of analyzing Model 2 by datasets created from Model 1
dat2Mod1	SimResult that saves the simulation of analyzing Model 1 by datasets created from Model 2
dat2Mod2	SimResult that saves the simulation of analyzing Model 2 by datasets created from Model 2
usedFit	Vector of names of fit indices that researchers wish to getCutoffs from. The default is to getCutoffs of all fit indices.
nVal	The sample size value that researchers wish to find the *p* value from.
pmMCARval	The percent missing completely at random value that researchers wish to find the *p* value from.
pmMARval	The percent missing at random value that researchers wish to find the the *p* value from.
df	The degree of freedom used in spline method in predicting the fit indices by the predictors. If df is 0, the spline method will not be applied.
onetailed	If TRUE, the function will convert the *p* value based on two-tailed test.

Details

In comparing fit indices, the *p* value is the proportion of the number of replications that provide less preference for either model 1 or model 2 than the analysis result from the observed data. In two-tailed test, the function will report the proportion of values under the sampling distribution that are more extreme that one obtained from real data. If the resulting `p` value is high (> .05) on one model and low (< .05) in the other model, the model with high `p` value is preferred. If the `p` values are both high or both low, the decision is undetermined.

Value

This function provides a vector of *p* values based on the comparison of the difference in fit indices from the real data with the simulation results. The *p* values of fit indices are provided, as well as two additional 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.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `SimResult` to run a simulation study

Examples

```
## Not run:
# Model A; Factor 1 --> Factor 2; Factor 2 --> Factor 3
library(lavaan)
loading <- matrix(0, 11, 3)
loading[1:3, 1] <- NA
loading[4:7, 2] <- NA
loading[8:11, 3] <- NA
path.A <- matrix(0, 3, 3)
path.A[2, 1] <- NA
path.A[3, 2] <- NA
model.A <- estmodel(LY=loading, BE=path.A, modelType="SEM", indLab=c(paste("x", 1:3, sep=""),
paste("y", 1:8, sep="")))

out.A <- analyze(model.A, PoliticalDemocracy)

# Model A; Factor 1 --> Factor 3; Factor 3 --> Factor 2
path.B <- matrix(0, 3, 3)
path.B[3, 1] <- NA
path.B[2, 3] <- NA
```

```
model.B <- estmodel(LY=loading, BE=path.B, modelType="SEM", indLab=c(paste("x", 1:3, sep=""),
paste("y", 1:8, sep="")))

out.B <- analyze(model.B, PoliticalDemocracy)

loading.mis <- matrix("runif(1, -0.2, 0.2)", 11, 3)
loading.mis[is.na(loading)] <- 0

# Create SimSem object for data generation and data analysis template
datamodel.A <- model.lavaan(out.A, std=TRUE, LY=loading.mis)
datamodel.B <- model.lavaan(out.B, std=TRUE, LY=loading.mis)

# Get sample size
n <- nrow(PoliticalDemocracy)

# The actual number of replications should be greater than 20.
output.A.A <- sim(20, n=n, model.A, generate=datamodel.A)
output.A.B <- sim(20, n=n, model.B, generate=datamodel.A)
output.B.A <- sim(20, n=n, model.A, generate=datamodel.B)
output.B.B <- sim(20, n=n, model.B, generate=datamodel.B)

# Find the p-value comparing the observed fit indices against the simulated
# sampling distribution of fit indices

pValueNonNested(out.A, out.B, output.A.A, output.A.B, output.B.A, output.B.B)

# If the p-value for model A is significant but the p-value for model B is not
# significant, model B is preferred.

## End(Not run)
```

---

rawDraw *Draw values from vector or matrix objects*

Description

Takes one matrix or vector object (SimMatrix or SimVector) and returns a matrix or a vector with numerical values for population parameters. If a matrix is symmetric, it is arbitrarily chosen that parameters on the upper triangular elements are set equal to the parameters on the lower triangular elements.

Usage

```
rawDraw(simDat, constraint = TRUE, misSpec = TRUE, parMisOnly = FALSE,
    misOnly = FALSE)
```

Arguments

simDat	A matrix or vector object (SimMatrix or SimVector)

`constraint`	If TRUE, then constraints are applied simultaneously
`misSpec`	If TRUE, then a list is returned with [[1]] parameters with no misspec and [[2]] same parameters + misspec (if any)
`parMisOnly`	If TRUE, then only the parameters + misspecification is returned
`misOnly`	If TRUE, then only the misspecification is returned

Value

A matrix (or vector) or a list of matrices (or vectors) which contains the draw result.

Author(s)

Patrick Miller (University of Notre Dame; `<pmille13@nd.edu>`)

Examples

```
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] <- "runif(1, 0.5, 0.7)"
loadingVal[4:6, 2] <- "runif(1, 0.5, 0.7)"
loadingVal[1:6, 3] <- "runif(1, 0.3, 0.5)"
loadingVal[7, 3] <- 1
loading.mis <- matrix("runif(1, -0.2, 0.2)", 7, 3)
loading.mis[is.na(loading)] <- 0
loading.mis[,3] <- 0
loading.mis[7,] <- 0
loading[1:3, 1] <- "con1"
LY <- bind(loading, loadingVal, misspec=loading.mis)

# Draw values
rawDraw(LY)

# Draw only model parameters containing misspecification
rawDraw(LY, parMisOnly=TRUE)

# Draw only misspecification.
rawDraw(LY, misOnly=TRUE)
```

`setPopulation` *Set the data generation population model underlying an object*

Description

This function will set the data generation population model to be an appropriate one. If the appropriate data generation model is specified, the additional features can be seen in `summary` or `summaryParam` functions on the target object, such as bias in parameter estimates or percentage coverage.

Usage

```
setPopulation(target, population)
```

Arguments

`target`	The result object that you wish to set the data generation population model (`linkS4class{SimResult}`).
`population`	The population parameters specified in the `linkS4class{SimSem}` object

Value

The target object that is changed the parameter.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `SimResult` for result object

Examples

```
# See each class for an example.
## Not run:
# Data generation model
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] <- "runif(1, 0.5, 0.7)"
loadingVal[4:6, 2] <- "runif(1, 0.5, 0.7)"
loadingVal[1:6, 3] <- "runif(1, 0.3, 0.5)"
loadingVal[7, 3] <- 1
loading.mis <- matrix("runif(1, -0.2, 0.2)", 7, 3)
loading.mis[is.na(loading)] <- 0
loading.mis[,3] <- 0
loading.mis[7,] <- 0
LY <- bind(loading, loadingVal, misspec=loading.mis)

RPS <- binds(diag(3))

path <- matrix(0, 3, 3)
path[2, 1] <- NA
BE <- bind(path, "runif(1, 0.3, 0.5)")

RTE <- binds(diag(7))

VY <- bind(c(rep(NA, 6), 0), c(rep(1, 6), ""))
```

```
datamodel <- model(LY=LY, RPS=RPS, BE=BE, RTE=RTE, VY=VY, modelType="SEM")

# Data analysis model
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)
analysis <- estmodel(LY=loading, BE=path, TE=errorCov, PS=facCov, modelType="SEM",
indLab=paste("y", 1:7, sep=""))

# In reality, more than 10 replications are needed.
Output <- sim(10, n=200, analysis, generate=datamodel)

# Population
loadingVal <- matrix(0, 7, 3)
loadingVal[1:3, 1] <- 0.6
loadingVal[4:6, 2] <- 0.6
loadingVal[7, 3] <- 1
LY <- bind(loading, loadingVal)
pathVal <- matrix(0, 3, 3)
pathVal[2, 1] <- 0.4
pathVal[1, 3] <- 0.4
pathVal[2, 3] <- 0.4
BE <- bind(path, pathVal)
PS <- binds(facCov)
errorCovVal <- diag(0.64, 7)
errorCovVal[7, 7] <- 0
TE <- binds(errorCov, errorCovVal)
population <- model(LY=LY, PS=PS, BE=BE, TE=TE, modelType="SEM")

# Set up the new population
Output2 <- setPopulation(Output, population)

# This summary will contain the bias information
summary(Output2)

## End(Not run)
```

---

sim *Run a monte carlo simulation with a structural equation model.*

Description

This function can be used to generate data, analyze the generated data, and summarized into a result object where parameter estimates, standard errors, fit indices, and other characteristics of each replications are saved.

Usage

```
sim(nRep, model, n, generate = NULL, ..., rawData = NULL, miss = NULL, datafun=NULL,
lavaanfun = "lavaan", outfun=NULL, outfundata = NULL, pmMCAR = NULL,
pmMAR = NULL,  facDist = NULL, indDist = NULL, errorDist = NULL,
sequential = FALSE, saveLatentVar = FALSE, modelBoot = FALSE, realData = NULL,
covData = NULL, maxDraw = 50, misfitType = "f0", misfitBounds = NULL,
averageNumMisspec = FALSE, optMisfit=NULL, optDraws = 50,
createOrder = c(1, 2, 3), aux = NULL, group = NULL, mxFit = FALSE,
mxMixture = FALSE, citype = NULL, cilevel = 0.95, seed = 123321, silent = FALSE,
multicore = options('simsem.multicore')[[1]], cluster = FALSE,
numProc = NULL, paramOnly = FALSE, dataOnly=FALSE, smartStart=FALSE,
previousSim = NULL, completeRep = FALSE, stopOnError = FALSE)
```

Arguments

nRep	Number of replications. If any of the n, pmMCAR, or pmMAR arguments are specified as lists, the number of replications will default to the length of the list(s), and nRep need not be specified.
model	There are three options for this argument: 1. SimSem object created by model, 2. lavaan script, lavaan parameter table, fitted lavaan object matching the analysis model, or a list that contains all argument that users use to run lavaan (including cfa, sem, lavaan), 3. MxModel object from the OpenMx package, or 4. a function that takes a data set and return a list of coef, se, and converged (see details below). For the SimSem object, if the generate argument is not specified, then the object in the model argument will be used for both data generation and analysis. If generate is specified, then the model argument will be used for data analysis only.
n	Sample size. Either a single value, or a list of values to vary sample size across replications. The n argument can also be specified as a random distribution object; if any resulting values are non-integers, the decimal will be rounded.
generate	There are four options for this argument: 1. SimSem object created by model, 2. lavaan script, lavaan parameter table (for data generation; see simulateData), fitted lavaan object that estimated all nonzero population parameters, or a list that contains all argument that users use to run simulateData, 3. MxModel object with population parameters specified in the starting values of all matrices in the model, 4. a function that take only one sample size argument (by integer for single-group model or by a vector of integers for multiple-group model). The generate argument cannot be specified the same time as the rawData argument.
rawData	There are two options for this argument: 1. a list of data frames to be used in simulations or 2. a population data. If a list of data frames is specified, the nRep and n arguments must not be specified. If a population data frame is specified, the nRep and n arguments are required.

`miss` A missing data template created using the `miss` function.

`datafun` A function to be applied to each generated data set across replications.

`lavaanfun` The character of the function name used in running lavaan model (`"cfa"`, `"sem"`, `"growth"`, `"lavaan"`). This argument is required only when lavaan script or a list of arguments is specified in the `model` argument.

`outfun` A function to be applied to the `lavaan` output at each replication. Output from this function in each replication will be saved in the simulation output (`SimResult`), and can be obtained using the `getExtraOutput` function.

`outfundata` A function to be applied to the `lavaan` output and the generated data at each replication. Users can get the characteristics of the generated data and also compare the characteristics with the generated output. The output from this function in each replication will be saved in the simulation output (`SimResult`), and can be obtained using the `getExtraOutput` function.

`pmMCAR` The percentage of data completely missing at random (0 <= pmMCAR < 1). Either a single value or a vector of values in order to vary pmMCAR across replications (with length equal to nRep or a divisor of nRep). The `objMissing` argument is only required when specifying complex missing value data generation, or when using multiple imputation.

`pmMAR` The percentage of data missing at random (0 <= pmCAR < 1). Either a single value or a vector of values in order to vary pmCAR across replications (with length equal to nRep or a divisor of nRep). The `objMissing` argument is only required when specifying complex missing value data generation, or when using multiple imputation.

`facDist` Factor distributions. Either a list of `SimDataDist` objects or a single `SimDataDist` object to give all factors the same distribution. Use when `sequential` is `TRUE`.

`indDist` Indicator distributions. Either a list of `SimDataDist` objects or a single `SimDataDist` object to give all indicators the same distribution. Use when `sequential` is `FALSE`.

`errorDist` An object or list of objects of type `SimDataDist` indicating the distribution of errors. If a single `SimDataDist` is specified, each error will be genrated with that distribution.

`sequential` If `TRUE`, a sequential method is used to generate data in which factor data is generated first, and is subsequently applied to a set of equations to obtain the indicator data. If `FALSE`, data is generated directly from model-implied mean and covariance of the indicators.

`saveLatentVar` If `TRUE`, the generated latent variable scores and measurement error scores are also provided as the attribute of the generated data. Users can use the `outfundata` to compare the latent variable scores with the estimated output. The `sequential` argument must be `TRUE` in order to use this option.

`modelBoot` When specified, a model-based bootstrap is used for data generation (for use with the `realData` argument). See `draw` for further information.

`realData` A data.frame containing real data. Generated data will follow the distribution of this data set.

covData	A data.frame containing covariate data, which can have any distributions. This argument is required when users specify GA or KA matrices in the model template (SimSem).
maxDraw	The maximum number of attempts to draw a valid set of parameters (no negative error variance, standardized coefficients over 1).
misfitType	Character vector indicating the fit measure used to assess the misfit of a set of parameters. Can be "f0", "rmsea", "srmr", or "all".
misfitBounds	Vector that contains upper and lower bounds of the misfit measure. Sets of parameters drawn that are not within these bounds are rejected.
averageNumMisspec	If TRUE, the provided fit will be divided by the number of misspecified parameters.
optMisfit	Character vector of either "min" or "max" indicating either maximum or minimum optimized misfit. If not null, the set of parameters out of the number of draws in "optDraws" that has either the maximum or minimum misfit of the given misfit type will be returned.
optDraws	Number of parameter sets to draw if optMisfit is not null. The set of parameters with the maximum or minimum misfit will be returned.
createOrder	The order of 1) applying equality/inequality constraints, 2) applying misspecification, and 3) fill unspecified parameters (e.g., residual variances when total variances are specified). The specification of this argument is a vector of different orders of 1 (constraint), 2 (misspecification), and 3 (filling parameters). For example, `c(1,2,3)` is to apply constraints first, then add the misspecification, and finally fill all parameters. See the example of how to use it in the draw function.
aux	The names of auxiliary variables saved in a vector.
group	The name of the group variable. This argument is used when lavaan script or MxModel is used in the model only.
mxFit	A logical whether to find an extensive list of fit measures (which will be slower). This argument is applicable when MxModel is used in the model argument only.
mxMixture	A logical whether to the analysis model is a mixture model. This argument is applicable when MxModel is used in the model argument only.
citype	Type of confidence interval. For the current version, this argument will be forwarded to the "boot.ci.type" argument in the parameterEstimates function from the lavaan package. This argument is not active when the OpenMx package is used.
cilevel	Confidence level. For the current version, this argument will be forwarded to the "level" argument in the parameterEstimates function from the lavaan package. This argument is not active when the OpenMx package is used.
seed	Random number seed. Note that the seed number is always fixed in the simsem so that users can always replicate the same simulatin or can be confidence that the same data set are generated. Reproducibility across multiple cores or clusters is ensured using R'Lecuyer package.
silent	If TRUE, suppress warnings.

multicore	Users may put TRUE or FALSE. If TRUE, multiple processors within a computer will be utilized. The default value is FALSE. Users may permanently change the default value by assigning the following line: options('simsem.multicore' = TRUE)
cluster	Not applicable now. Used to specify nodes in hpc in order to be parallelizable.
numProc	Number of processors for using multiple processors. If it is NULL, the package will find the maximum number of processors.
paramOnly	If TRUE, only the parameters from each replication will be returned.
dataOnly	If TRUE, only the raw data generated from each replication will be returned.
smartStart	Defaults to FALSE. If TRUE, population parameter values that are real numbers will be used as starting values. When tested in small models, the time elapsed when using population values as starting values was greater than the time reduced during analysis, and convergence rates were not affected.
previousSim	A result object that users wish to add the results of the current simulation in
completeRep	If TRUE, the function will run until the number of convergent replication equal to the specified nRep.
stopOnError	If TRUE, stop running the simulation when the error occurs during the data analysis on any replications.
...	Additional arguments to be passed to lavaan. See also lavOptions

Details

This function is executed as follows: 1. parameters are drawn from the specified data-generation model (applicable only simsem model template, SimSem, only), 2. the drawn (or the specified) parameters are used to create data, 3. data can be transformed using the datafun argument, 4. specified missingness (if any) is imposed, 5. data are analyzed using the specified analysis model, 6. parameter estimates, standard errors, fit indices, and other characteristics of a replication are extracted, 7. additional outputs (if any) are extracted using the outfun argument, and 8. results across replications are summarized in a result object, SimResult).

There are six ways to provide or generate data in this function:

1. SimSem can be used as a template to generate data, which can be created by the model function. The SimSem can be specified in the generate argument.
2. lavaan script, parameter table for the lavaan package, or a list of arguments for the simulateData function. The lavaan script can be specified in the generate argument.
3. MxModel object from the OpenMx package. The MxModel object can be specified in the generate argument.
4. A list of raw data for each replication can be provided for the rawData argument. The sim function will analyze each data and summarize the result. Note that the generate, n and nRep could not be specified if the list of raw data is provided.
5. Population data can be provided for the rawData argument. The sim function will randomly draw sample data sets and analyze data. Note that the n and nRep must be specified if the population data are provided. The generate argument must not be specified.

6. A function can be used to generate data. The function must take sample size in a numeric format (or a vector of numerics for multiple groups) and return a data frame of the generated data set. Note that parameter values and their standardized values can be provided by using the attributes of the resulting data set. That is, users can assign parameter values and standardized parameter values to `attr(data,"param")` and `attr(data,"stdparam")`.

Note that all generated or provided data can be transformed based on Bollen-Stine approach by providing a real data in the `realData` argument if any of the first three methods are used.

There are four ways to analyze the data sets for each replication by setting the `model` argument as

1. `SimSem` can be used as a template for data analysis.
2. `lavaan` script, parameter table for the `lavaan` package, or a list of arguments for the `lavaan`, `sem`, `cfa`, or `growth` function. Note that if the desired function to analyze data can be specified in the `lavaanfun` argument, which the default is the `lavaan` function
3. `MxModel` object from the `OpenMx` package. The object does not need to have data inside. Note that if users need an extensive fit indices, the `mxFit` argument should be specified as `TRUE`. If users wish to analyze by mixture model, the `mxMixture` argument should be `TRUE` such that the `sim` function knows how to handle the data.
4. A function that takes a data set and returns a list. The list must contain at least three objects: a vector of parameter estimates (`coef`), a vector of standard error (`se`), and the convergence status as `TRUE` or `FALSE` (`converged`). There are seven optional objects in the list: a vector of fit indices (`fit`), a vector of standardized estimates (`std`), a vector of standard errors of standardized estimates (`stdse`), fraction missing type I (`FMI1`), fraction missing type II (`FMI2`), lower bounds of confidence intervals (`cilower`), and upper bounds of confidence intervals (`ciupper`). Note that the `coef`, `se`, `std`, `stdse`, `FMI1`, `FMI2`, `cilower`, and `ciupper` must be a vector with names. The name of those vectors across different objects must be the same. Users may optionally specify other objects in the list; however, the results of the other objects will not be automatically combined. Users need to specify the `outfun` argument to get the extra objects. For example, researchers may specify `residuals` in the list. The outfun argument should have the function as follows: `function(obj) obj$residuals`.

Any combination of data-generation methods and data-analysis methods are valid. For example, data can be simulated using lavaan script and analyzed by `MxModel`. Paralleled processing can be enabled using the `multicore` argument.

Value

A result object (`SimResult`)

Author(s)

Patrick Miller (University of Notre Dame; <pmille13@nd.edu>) Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `SimResult` for the resulting output description

Examples

```
# Please go to www.simsem.org for more examples.

# Example of using simsem model template
library(lavaan)

loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
LY <- bind(loading, 0.7)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))

VY <- bind(rep(NA,6),2)

CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType = "CFA")

# In reality, more than 5 replications are needed.
Output <- sim(5, CFA.Model, n=200)
summary(Output)

# Example of using simsem model template

popModel <- "
f1 =~ 0.7*y1 + 0.7*y2 + 0.7*y3
f2 =~ 0.7*y4 + 0.7*y5 + 0.7*y6
f1 ~~ 1*f1
f2 ~~ 1*f2
f1 ~~ 0.5*f2
y1 ~~ 0.49*y1
y2 ~~ 0.49*y2
y3 ~~ 0.49*y3
y4 ~~ 0.49*y4
y5 ~~ 0.49*y5
y6 ~~ 0.49*y6
"

analysisModel <- "
f1 =~ y1 + y2 + y3
f2 =~ y4 + y5 + y6
"

Output <- sim(5, model=analysisModel, n=200, generate=popModel, std.lv=TRUE, lavaanfun = "cfa")
summary(Output)

# Example of using population data

pop <- data.frame(y1 = rnorm(100000, 0, 1), y2 = rnorm(100000, 0, 1))
```

```
covModel <- "
y1 ~~ y2
"

Output <- sim(5, model=covModel, n=200, rawData=pop, lavaanfun = "cfa")
summary(Output)

# Example of data transformation: Transforming to standard score
fun1 <- function(data) {
temp <- scale(data)
as.data.frame(temp)
}

# Example of additional output: Extract modification indices from lavaan
fun2 <- function(out) {
inspect(out, "mi")
}

# In reality, more than 5 replications are needed.
Output <- sim(5, CFA.Model,n=200,datafun=fun1, outfun=fun2)
summary(Output)

# Get modification indices
getExtraOutput(Output)

# Example of additional output: Comparing latent variable correlation

outfundata <- function(out, data) {
predictcor <- inspect(out, "coef")$psi[2, 1]
latentvar <- attr(data, "latentVar")[,c("f1", "f2")]
latentcor <- cor(latentvar)[2,1]
latentcor - predictcor
}

Output <- sim(5, CFA.Model,n=200, sequential = TRUE, saveLatentVar = TRUE,
outfundata = outfundata)

getExtraOutput(Output)

# Example of analyze using a function

analyzeFUN <- function(data) {
out <- lm(y2 ~ y1, data=data)
coef <- coef(out)
se <- sqrt(diag(vcov(out)))
fit <- c(loglik = as.numeric(logLik(out)))
converged <- TRUE # Assume to be convergent all the time
return(list(coef = coef, se = se, fit = fit, converged = converged))
}

Output <- sim(5, model=analyzeFUN, n=200, rawData=pop, lavaanfun = "cfa")
summary(Output)
```

SimDataDist-class *Class "SimDataDist": Data distribution object*

Description

This class will provide the distribution of a dataset.

Objects from the Class

Objects can be created by `bindDist` function. It can also be called from the form `new("SimDataDist",...)`.

Slots

`p`: Number of variables

`margins`: A character vector specifying all the marginal distributions

`paramMargins`: A list whose each component is a list of named components, giving the parameter values of the marginal distributions.

`keepScale`: Transform back to retain the mean and standard deviation of a variable equal to the model implied mean and standard deviation (with sampling error)

`reverse`: To mirror each variable or not. If `TRUE`, reverse the distribution of a variable (e.g., from positive skewed to negative skewed).

`copula`: The multivariate copula template for data generation. See `bindDist`

`skewness`: The target skewness values of each variable

`kurtosis`: The target (excessive) kurtosis values of each variable

Methods

- summaryTo summarize the object
- plotDistTo plot a density distribution (for one variable) or a contour plot (for two variables).

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `bindDist` The constructor of this class.

Examples

```
showClass("SimDataDist")

d1 <- list(df=2)
d2 <- list(df=3)
d3 <- list(df=4)
d4 <- list(df=5)
```

```
d5 <- list(df=3)
d6 <- list(df=4)
d7 <- list(df=5)
d8 <- list(df=6)

dist <- bindDist(c(rep("t", 4), rep("chisq", 8)), d1, d2, d3, d4, d5, d6, d7, d8, d5, d6, d7, d8)
summary(dist)

dist2 <- bindDist(skewness = seq(-3, 3, length.out=12), kurtosis = seq(2, 5, length.out=12))
summary(dist2)
```

SimMatrix-class *Matrix object: Random parameters matrix*

Description

This object can be used to represent a matrix in SEM model. It contains free parameters, fixed values, starting values, and model misspecification. This object can be represented mean, intercept, or variance vectors.

Objects from the Class

This object is created by `bind` or `binds` function.

Slots

`free`: The free-parameter vector. Any NA elements or character elements are free. Any numeric elements are fixed as the specified number. If any free elements have the same characters (except NA), the elements are equally constrained.

`popParam`: Real population parameters of the free elements.

`misspec`: Model misspecification that will be added on top of the fixed and real parameters.

`symmetric`: If TRUE, the specified matrix is symmetric.

Methods

`rawDraw` Draws data-generation parameters.

`summaryShort` Provides a short summary of all information in the object

`summary` Provides a thorough description of all information in the object

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- `SimVector` for random parameter vector.

Examples

```
showClass("SimMatrix")

loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loadingValues <- matrix(0, 6, 2)
loadingValues[1:3, 1] <- 0.7
loadingValues[4:6, 2] <- 0.7
LY <- bind(loading, loadingValues)
summary(LY)
rawDraw(LY)

LY <- bind(loading, "rnorm(1, 0.6, 0.05)")
summary(LY)
rawDraw(LY)

mis <- matrix("runif(1, -0.1, 0.1)", 6, 2)
mis[is.na(loading)] <- 0
LY <- bind(loading, "rnorm(1, 0.6, 0.05)", mis)
summary(LY)
rawDraw(LY)
```

SimMissing-class *Class* "SimMissing"

Description

Missing information imposing on the complete dataset

Objects from the Class

Objects can be created by `miss` function.

Slots

`cov`: Column indices of any normally distributed covariates used in the data set.

`pmMCAR`: Decimal percent of missingness to introduce completely at random on all variables.

`pmMAR`: Decimal percent of missingness to introduce using the listed covariates as predictors.

`logit`: The script used for imposing missing values by logistic regression. See `miss` for further details.

`nforms`: The number of forms for planned missing data designs, not including the shared form.

`itemGroups`: List of lists of item groupings for planned missing data forms. Without this, items will be divided into groups sequentially (e.g. 1-3,4-6,7-9,10-12)

`twoMethod`: Vector of (percent missing, column index). Will put a given percent missing on that column in the matrix to simulate a two method planned missing data research design.

prAttr: Probability (or vector of probabilities) of an entire case being removed due to attrition at a given time point. See imposeMissing for further details.

m: The number of imputations. The default is 0 such that the full information maximum likelihood is used.

package: The package to be used in multiple imputation. The default value of this function is "default". For the default option, if m is 0, the full information maximum likelihood is used. If m is greater than 0, the mice package is used.

convergentCutoff: If the proportion of convergent results across imputations are greater than the specified value (the default is 80%), the analysis on the dataset is considered as convergent. Otherwise, the analysis is considered as nonconvergent. This attribute is applied for multiple imputation only.

timePoints: Number of timepoints items were measured over. For longitudinal data, planned missing designs will be implemented within each timepoint.

ignoreCols: The columns not imposed any missing values for any missing data patterns

threshold: The threshold of covariates that divide between the area to impose missing and the area not to impose missing. The default threshold is the mean of the covariate.

covAsAux: If TRUE, the covariate listed in the object will be used as auxiliary variables when putting in the model object. If FALSE, the covariate will be included in the analysis.

logical: A matrix of logical values (TRUE/FALSE). If a value in the dataset is corresponding to the TRUE in the logical matrix, the value will be missing.

args: A list of additional options to be passed to the multiple impuatation function in each package.

Methods

- summary To summarize the object
- impose To impose missing information into data

Author(s)

Patrick Miller (University of Notre Dame; <pmille13@nd.edu>) Alexander M. Schoemann (East Carolina University; <schoemanna@ecu.edu>) Kyle Lang (University of Kansas; <kylelang@ku.edu>) Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- imposeMissing for directly imposing missingness into a dataset.

Examples

```
misstemplate <- miss(pmMCAR=0.2)
summary(misstemplate)
```

SimResult-class *Class* "SimResult"*: Simulation Result Object*

Description

This class will save data analysis results from multiple replications, such as fit indices cutoffs or power, parameter values, model misspecification, etc.

Objects from the Class

Objects can be created by `sim`.

Slots

`modelType`: Analysis model type (CFA, Path, or SEM)

`nRep`: Number of replications have been created and run simulated data.

`coef`: Parameter estimates from each replication

`se`: Standard errors of parameter estimates from each replication

`fit`: Fit Indices values from each replication

`converged`: The convergence status of each replication: 0 = convergent, 1 = not convergent, 2 = nonconvergent in multiple imputed results, 3 = improper solutions for SE (less than 0 or NA), 4 = improper solution for latent covariance matrix (Heywood case), 5 = improper solution for observed covariance matrix (Heywood case). For multiple imputations, these codes are applied when the proporion of imputed data sets with that characteristic is below the `convergentCutoff` threshold (see `linkS4class{SimMissing}`).

`seed`: Seed number.

`paramValue`: Population model underlying each simulated dataset.

`stdParamValue`: Standardized parameters of the population model underlying each simulated dataset.

`paramOnly`: If `TRUE`, the result object saves only population characteristics and do not save sample characteristics (e.g., parameter estimates and standard errors.

`misspecValue`: Misspecified-parameter values that are imposed on the population model in each replication.

`popFit`: The amount of population misfit. See details at `summaryMisspec`

`FMI1`: Fraction Missing Method 1.

`FMI2`: Fraction Missing Method 2.

`cilower`: Lower bounds of confidence interval.

`ciupper`: Upper bounds of confidence interval.

`stdCoef`: Standardized coefficients from each replication

`stdSe`: Standard Errors of Standardized coefficients from each replication

`n`: The total sample size of the analyzed data.

`nobs`: The sample size within each group.

`pmMCAR`: Percent missing completely at random.

`pmMAR`: Percent missing at random.

`extraOut`: Extra outputs obtained from running the function specified in `outfun` argument in the `sim` function.

`timing`: Time elapsed in each phase of the simulation.

Methods

The following methods are listed alphabetically. More details can be found by following the link of each method.

- `anova` to find the averages of model fit statistics and indices for nested models, as well as the differences of model fit indices among models. This function requires at least two `SimResult` objects.
- `coef` to extract parameter estimates of each replication
- `findCoverage` to find a value of independent variables (e.g., sample size) that provides a given value of coverage rate.
- `findPower` to find a value of independent variables (e.g., sample size) that provides a given value of power of a parameter estimate.
- `getCoverage` to get the coverage rate of the confidence interval of each parameter estimate
- `getCIwidth` to get a median or percentile rank (assurance) of confidence interval widths of parameters estimates
- `getCutoff` to get the cutoff of fit indices based on a priori alpha level.
- `getCutoffNested` to get the cutoff of the difference in fit indices of nested models based on a priori alpha level.
- `getCutoffNonNested` to get the cutoff of the difference in fit indices of nonnested models based on a priori alpha level.
- `getExtraOutput` to get extra outputs that users requested before running a simulation
- `getPopulation` to get population parameter values underlying each dataset
- `getPower` to get the power of each parameter estimate
- `getPowerFit` to get the power in rejecting alternative models based on absolute model fit cutoff.
- `getPowerFitNested` to get the power in rejecting alternative models based on the difference between model fit cutoffs of nested models.
- `getPowerFitNonNested` to get the power in rejecting alternative models based on the difference between model fit cutoffs of nonnested models.
- `inspect` Extract target information from the simulation result. The available information is listed in this `link`
- `likRatioFit` to find the likelihood ratio (or Bayes factor) based on the bivariate distribution of fit indices
- `plotCoverage` to plot the coverage rate of confidence interval of parameter estimates
- `plotCIwidth` to plot confidence interval widths with a line of a median or percentile rank (assurance)

- plotCutoff to plot sampling distributions of fit indices with an option to draw fit indices cutoffs by specifying a priori alpha level.
- plotCutoffNested to plot sampling distributions of the difference in fit indices between nested models with an option to draw fit indices cutoffs by specifying a priori alpha level.
- plotCutoffNonNested to plot sampling distributions of the difference in fit indices between nonnested models with an option to draw fit indices cutoffs by specifying a priori alpha level.
- plotMisfit to visualize the population misfit and misspecified parameter values
- plotPower to plot power of parameter estimates
- plotPowerFit to plot the power in rejecting alternative models based on absolute model fit cutoff.
- plotPowerFitNested to plot the power in rejecting alternative models based on the difference between model fit cutoffs of nested models.
- plotPowerFitNonNested to plot the power in rejecting alternative models based on the difference between model fit cutoffs of nonnested models.
- pValue to find a p-value in comparing sample fit indices with the null sampling distribution of fit indices
- pValueNested to find a p-value in comparing the difference in sample fit indices between nested models with the null sampling distribution of the difference in fit indices
- pValueNonNested to find a p-value in comparing the difference in sample fit indices between nonnested models with the null sampling distribution of the difference in fit indices
- setPopulation to set population model for computing bias
- summary to summarize the result output
- summaryConverge to provide a head-to-head comparison between the characteristics of convergent and nonconvergent replications
- summaryMisspec to provide a summary of model misfit
- summaryParam to summarize all parameter estimates
- summaryPopulation to summarize the data generation population underlying the simulation study.
- summarySeed to provide a summary of the seed number in the simulation
- summaryShort to provide a short summary of the result output
- summaryTime to provide a summary of time elapsed in the simulation

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- sim for the constructor of this class

Examples

```
showClass("SimResult")
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, 0.7)
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output <- sim(5, n=500, CFA.Model)

# Summary the simulation result
summary(Output)

# Short summary of the simulation result
summaryShort(Output)

# Find the fit index cutoff
getCutoff(Output, 0.05)

# Summary of parameter estimates
summaryParam(Output)

# Summary of population parameters
summaryPopulation(Output)
```

SimSem-class *Class* "SimSem"

Description

The template containing data-generation and data-analysis specification

Objects from the Class

Objects can be created by `model`.

Slots

`pt`**:** Parameter table used in data analysis

`dgen`**:** Data generation template

`modelType`**:** Type of models (CFA, Path, or SEM) contained in this object

`groupLab`**:** The label of grouping variable

`con`**:** The list of defined parameters, equality constraints, or inequality constraints specified in the model

Methods

summary Get the summary of model specification

Author(s)

Patrick Miller (University of Notre Dame; <pmille13@nd.edu>), Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

- Create an object this class by CFA, Path Analysis, or SEM model by `model`.

Examples

```
showClass("SimSem")

loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loadingValues <- matrix(0, 6, 2)
loadingValues[1:3, 1] <- 0.7
loadingValues[4:6, 2] <- 0.7
LY <- bind(loading, loadingValues)
summary(LY)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

# Error Correlation Object
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTE <- binds(error.cor)

CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")
summary(CFA.Model)
```

`SimVector-class` *Vector object: Random parameters vector*

Description

This object can be used to represent a vector in SEM model. It contains free parameters, fixed values, starting values, and model misspecification. This object can be represented mean, intercept, or variance vectors.

Objects from the Class

This object is created by `bind` function.

Slots

- `free`: The free-parameter vector. Any NA elements or character elements are free. Any numeric elements are fixed as the specified number. If any free elements have the same characters (except NA), the elements are equally constrained.
- `popParam`: Real population parameters of the free elements.
- `misspec`: Model misspecification that will be added on top of the fixed and real parameters.

Methods

- `rawDraw` Draws data-generation parameters.
- `summaryShort` Provides a short summary of all information in the object
- `summary` Provides a thorough description of all information in the object

Author(s)

Patrick Miller (University of Notre Dame; <pmille13@nd.edu>) Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

`SimMatrix` for random parameter matrix

Examples

```
showClass("SimVector")

factor.mean <- rep(NA, 2)
factor.mean.starting <- c(5, 2)
AL <- bind(factor.mean, factor.mean.starting)
rawDraw(AL)
summary(AL)
summaryShort(AL)
```

---

summaryConverge — *Provide a comparison between the characteristics of convergent replications and nonconvergent replications*

Description

This function provides a comparison between the characteristics of convergent replications and nonconvergent replications. The comparison includes sample size (if varying), percent missing completely at random (if varying), percent missing at random (if varying), parameter values, misspecified-parameter values (if applicable), and population misfit (if applicable).

Usage

```
summaryConverge(object, std = FALSE, improper = TRUE)
```

Arguments

object	SimResult object being described
std	If TRUE, the standardized parameter values are used instead of unstandardized parameter values.
improper	If TRUE, include the replications that provided improper solutions

Value

A list with the following elements:

- Converged The number of convergent and nonconvergent replications
- n Sample size
- pmMCAR Percent missing completely at random
- pmMAR Percent missing at random
- paramValue Parameter values
- misspecValue Misspecified-parameter values
- popFit Population misfit. See details of each element at summaryMisspec.

Each element will provide the head-to-head comparison between convergent and nonconvergent replications properties.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

Examples

```
## Not run:
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] <- "runif(1, 0.3, 0.5)"
starting.BE[4, 3] <- "runif(1, 0.5, 0.7)"
mis.path.BE <- matrix(0, 4, 4)
mis.path.BE[4, 1:2] <- "runif(1, -0.1, 0.1)"
BE <- bind(path.BE, starting.BE, misspec=mis.path.BE)

residual.error <- diag(4)
residual.error[1,2] <- residual.error[2,1] <- NA
RPS <- binds(residual.error, "rnorm(1, 0.3, 0.1)")

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
mis.loading <- matrix("runif(1, -0.3, 0.3)", 12, 4)
mis.loading[is.na(loading)] <- 0
```

```
LY <- bind(loading, "runif(1, 0.7, 0.9)", misspec=mis.loading)

mis.error.cor <- matrix("rnorm(1, 0, 0.1)", 12, 12)
diag(mis.error.cor) <- 0
RTE <- binds(diag(12), misspec=mis.error.cor)

SEM.Model <- model(RPS = RPS, BE = BE, LY=LY, RTE=RTE, modelType="SEM")

n1 <- list(mean = 0, sd = 0.1)
chi5 <- list(df = 5)

facDist <- bindDist(c("chisq", "chisq", "norm", "norm"), chi5, chi5, n1, n1)

# In reality, more than 50 replications are needed.
simOut <- sim(50, n=500, SEM.Model, sequential=TRUE, facDist=facDist, estimator="mlr")

# Summary the convergent and nonconvergent replications
summaryConverge(simOut)

## End(Not run)
```

---

summaryFit *Provide summary of model fit across replications*

Description

This function will provide fit index cutoffs for values of alpha, and mean fit index values across all replications.

Usage

```
summaryFit(object, alpha = NULL, improper = TRUE, usedFit = NULL)
```

Arguments

object	SimResult to be summarized
alpha	The alpha level used to find the fit indices cutoff. If there is no varying condition, a vector of different alpha levels can be provided.
improper	If TRUE, include the replications that provided improper solutions
usedFit	Vector of names of fit indices that researchers wish to summarize.

Value

A data frame that provides fit statistics cutoffs and means

When linkS4class{SimResult} has fixed simulation parameters the first colmns are fit index cutoffs for values of alpha and the last column is the mean fit across all replications. Rows are

- Chi Chi-square fit statistic

- AIC Akaike Information Criterion
- BIC Baysian Information Criterion
- RMSEA Root Mean Square Error of Approximation
- CFI Comparative Fit Index
- TLI Tucker-Lewis Index
- SRMR Standardized Root Mean Residual

When `linkS4class{SimResult}` has random simulation parameters (sample size or percent missing), columns are the fit indices listed above and rows are values of the random parameter.

Author(s)

Alexander M. Schoemann (East Carolina University; <schoemanna@ecu.edu>) Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

`SimResult` for the result object input

Examples

```
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, 0.7)
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output <- sim(5, n=500, CFA.Model)

# Summarize the sample fit indices
summaryFit(Output)
```

`summaryMisspec` *Provide summary of the population misfit and misspecified-parameter values across replications*

Description

This function provides the summary of the population misfit and misspecified-parameter values across replications. The summary will be summarized for the convergent replications only.

Usage

```
summaryMisspec(object, improper = TRUE)
```

Arguments

- `object` — `SimResult` object being described
- `improper` — If TRUE, include the replications that provided improper solutions

Value

A data frame that provides the summary of population misfit and misspecified-parameter values imposed on the real parameters.

The discrepancy value (f_0; Browne & Cudeck, 1992) is calculated by

$$F_0 = tr\left(\tilde{\Sigma}\Sigma^{-1}\right) - \log\left|\tilde{\Sigma}\Sigma^{-1}\right| - p + \left(\tilde{\mu} - \mu\right)' \Sigma^{-1} \left(\tilde{\mu} - \mu\right).$$

where μ is the model-implied mean from the real parameters, Σ is the model-implied covariance matrix from the real parameters, $\tilde{\mu}$ is the model-implied mean from the real and misspecified parameters, $\tilde{\Sigma}$ is the model-implied covariance matrix from the real and misspecified parameter, p is the number of indicators. For the multiple groups, the resulting f_0 value is the sum of this value across groups.

The root mean squared error of approximation (rmsea) is calculated by

$$rmsea = \sqrt{\frac{f_0}{df}}$$

where df is the degree of freedom in the real model.

The standardized root mean squared residual (srmr) can be calculated by

$$srmr = \sqrt{\frac{2 \sum_g \sum_i \sum_{j \le i} \left(\frac{s_{gij}}{\sqrt{s_{gii}}\sqrt{s_{gjj}}} - \frac{\hat{\sigma}_{gij}}{\sqrt{\hat{\sigma}_{gii}}\sqrt{\hat{\sigma}_{gjj}}} \right)}{g \times p(p+1)}}$$

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

Author(s)

Sunthud Pornprasertmanit (`<psunthud@gmail.com>`)

References

Browne, M. W., & Cudeck, R. (1992). Alternative ways of assessing model fit. *Sociological Methods & Research, 21*, 230-258.

See Also

`SimResult` for the object input

Examples

```
## Not run:
path <- matrix(0, 4, 4)
path[3, 1:2] <- NA
path[4, 3] <- NA
pathVal <- matrix("", 4, 4)
pathVal[3, 1:2] <- "runif(1, 0.3, 0.5)"
pathVal[4, 3] <- "runif(1, 0.5, 0.7)"
pathMis <- matrix(0, 4, 4)
pathMis[4, 1:2] <- "runif(1, -0.1, 0.1)"
BE <- bind(path, pathVal, pathMis)

residual.error <- diag(4)
residual.error[1,2] <- residual.error[2,1] <- NA
RPS <- binds(residual.error, "rnorm(1, 0.3, 0.1)")

Path.Model <- model(RPS = RPS, BE = BE, modelType="Path")

# The number of replications in actual analysis should be much more than 5
ParamObject <- sim(5, n=200, Path.Model)

# Summarize the model misspecification that is specified in the 'pathMis' object
summaryMisspec(ParamObject)

## End(Not run)
```

---

summaryParam — *Provide summary of parameter estimates and standard error across replications*

Description

This function will provide averages of parameter estimates, standard deviations of parameter estimates, averages of standard errors, and power of rejection with a priori alpha level for the null hypothesis of parameters equal 0.

Usage

```
summaryParam(object, alpha = 0.05, std = FALSE, detail = FALSE,
    improper = TRUE, digits = NULL, matchParam = FALSE)
```

Arguments

`object`	`SimResult` object being described
`alpha`	The alpha level used to find the statistical power of each parameter estimate
`std`	If `TRUE`, (a) standardized coefficients and their standard errors substitute unstandardized coefficients, (b) standardized parameter values substitute parameter values, (c) confidence intervals of standardized coefficients are calculated

using Wald confidence interval, and (d) all results (e.g., biases or coverage) are calculated based on standardized coefficients.

detail
: If TRUE, more details about each parameter estimate are provided, such as relative bias, standardized bias, or relative standard error bias.

improper
: If TRUE, include the replications that provided improper solutions

digits
: The number of digits rounded in the result. If `NULL`, the results will not be rounded.

matchParam
: If `TRUE`, only parameter estimates that have the same names as the parameter values will be reported. This argument is recommended when users know that the data-generation model and analysis model are the same. Then the comparison between the parameter estimates and parameter value will be valid.

Value

A data frame that provides the statistics described above from all parameters. For using with `linkS4class{SimResult}`, each column means

- `Estimate.Average`: Average of parameter estimates across all replications
- `Estimate.SD`: Standard Deviation of parameter estimates across all replications
- `Average.SE`: Average of standard errors across all replications
- `Power (Not equal 0)`: Proportion of significant replications when testing whether the parameters are different from zero. The alpha level can be set by the `alpha` argument of this function.
- `Average.Param`: Parameter values or average values of parameters if random parameters are specified
- `SD.Param`: Standard Deviations of parameters. Show only when random parameters are specified.
- `Average.Bias`: The difference between parameter estimates and parameter underlying data
- `SD.Bias`: Standard Deviations of bias across all replications. Show only when random parameters are specified. This value is the expected value of average standard error when random parameter are specified.
- `Coverage`: The percentage of (1-alpha)% confidence interval covers parameters underlying the data.
- `Rel.Bias`: Relative Bias, which is (`Estimate.Average` - `Average.Param`)/`Average.Param`. Hoogland and Boomsma (1998) proposed that the cutoff of .05 may be used for acceptable relative bias. This option will be available when `detail=TRUE`. This value will not be available when parameter values are very close to 0.
- `Std.Bias`: Standardized Bias, which is (`Estimate.Average` - `Average.Param`)/`Estimate.SD` for fixed parameters and (`Estimate.Average` - `Average.Param`)/`SD.Bias` for random parameters. Collins, Schafer, and Kam (2001) recommended that biases will be only noticeable when standardized bias is greater than 0.4 in magnitude. This option will be available when `detail=TRUE`
- `Rel.SE.Bias`: Relative Bias in standard error, which is (`Average.SE` - `Estimate.SD`)/`Estimate.SD` for fixed parameters and (`Average.SE` - `SD.Bias`)/`SD.Bias` for random parameters. Hoogland and Boomsma (1998) proposed that 0.10 is the acceptable level. This option will be available when `detail=TRUE`

- `Not Cover Below`: The percentage of (1-alpha)% confidence interval does not cover the parameter and the parameter is below the confidence interval.
- `Not Cover Above`: The percentage of (1-alpha)% confidence interval does not cover the parameter and the parameter is above the confidence interval.
- `Average CI Width`: The average of (1-alpha)% confidence interval width across replications.
- `SD CI Width`: The standard deviation of (1-alpha)% confidence interval width across replications.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

References

Collins, L. M., Schafer, J. L., & Kam, C. M. (2001). A comparison of inclusive and restrictive strategies in modern missing data procedures. *Psychological Methods, 6*, 330-351.

Hoogland, J. J., & Boomsma, A. (1998). Robustness studies in covariance structure modeling. *Sociological Methods & Research, 26*, 329-367.

See Also

`SimResult` for the object input

Examples

```
showClass("SimResult")
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, 0.7)
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output <- sim(5, n=500, CFA.Model)

# Summary of the parameter estimates
summaryParam(Output)

# Summary of the parameter estimates with additional details
summaryParam(Output, detail=TRUE)
```

`summaryPopulation` *Summarize the population model used for data generation underlying a result object*

Description

Summarize the population model used for data generation underlying a result object

Usage

```
summaryPopulation(object, std = FALSE, improper = TRUE)
```

Arguments

`object`	The result object that you wish to extract the data generation population model from (`linkS4class{SimResult}`).
`std`	If `TRUE`, the standardized parameter values are used instead of unstandardized parameter values.
`improper`	If TRUE, include the replications that provided improper solutions

Value

A `data.frame` contianing the summary of population model across replications.

Author(s)

Sunthud Pornprasertmanit (`<psunthud@gmail.com>`)

See Also

- `SimResult` for result object

Examples

```
## Not run:
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LY <- bind(loading, "runif(1, 0.4, 0.9)")
RPS <- binds(diag(1))
RTE <- binds(diag(6))
CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType="CFA")

# We will use only 10 replications to save time.
# In reality, more replications are needed.
Output <- sim(10, n=200, model=CFA.Model)

# Get the summary of population model
summaryPopulation(Output)
```

```
## End(Not run)
```

summarySeed *Summary of a seed number*

Description

Summary of a seed number used in the simulation

Usage

```
summarySeed(object)
```

Arguments

object	SimResult object being described

Value

The first section is the seed number used in running the whole simulation. The second section is the L'Ecuyer seed of the last replication.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

Examples

```
## Not run:
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
LY <- bind(loading, 0.7)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))

VY <- bind(rep(NA,6),2)

CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType = "CFA")

# In reality, more than 5 replications are needed.
Output <- sim(5, CFA.Model, n=200)
summarySeed(Output)

## End(Not run)
```

`summaryShort` *Provide short summary of an object.*

Description

Provide short summary if it is available. Otherwise, it is an alias for `summary`.

Usage

```
summaryShort(object, ...)
```

Arguments

`object`	Desired object being described
`...`	any additional arguments

Value

NONE. This function will print on screen only.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

See Also

This is the list of classes that can use `summaryShort` method.

- `SimMatrix`
- `SimVector`

Examples

```
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loadingValues <- matrix(0, 6, 2)
LY <- bind(loading, "runif(1, 0.8, 0.9)")
summaryShort(LY)
```

summaryTime *Time summary*

Description

Provide a summary of time elapsed in running the simulation.

Usage

```
summaryTime(object, units = "seconds")
```

Arguments

object	SimResult object being described
units	The units of time, which can be "seconds", "minutes", "hours", or "days"

Value

The first section is the actual time used in each step of the simulation. The second section is the average system (processor) time used in each replication. The third section is the summary of starting time, end time, total actual time, and total system time.

Author(s)

Sunthud Pornprasertmanit (<psunthud@gmail.com>)

Examples

```
## Not run:
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
LY <- bind(loading, 0.7)

latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPS <- binds(latent.cor, 0.5)

RTE <- binds(diag(6))

VY <- bind(rep(NA,6),2)

CFA.Model <- model(LY = LY, RPS = RPS, RTE = RTE, modelType = "CFA")

# In reality, more than 5 replications are needed.
Output <- sim(5, CFA.Model, n=200)
summaryTime(Output)

## End(Not run)
```

Index