Package 'simsem'

August 29, 2012

Type Package

Title SIMulated Structural Equation Modeling.

Version 0.3-0

Date 2012-08-27

Author Sunthud Pornprasertmanit <psunthud@ku.edu>, Patrick Miller <patr1ckm@ku.edu>, Alexander Schoemann <schoemann@ku.edu>

Maintainer Sunthud Pornprasertmanit <psunthud@ku.edu>

Depends R(>= 2.14), methods, lavaan, MASS

Suggests parallel, Amelia, copula, quantreg, splines, foreign,KernSmooth

Description This package can be used to generate data using the structural equation modeling framework. This package is tailored to use those simulated data for various purposes, such as model fit evaluation, power analysis, or missing data handling and planning.

License GPL (>= 2)

LazyLoad yes

URL http://www.simsem.org

R topics documented:

`analyze` *TBA*

Description

TBA

Usage

```
analyze(model, data, package="lavaan", simMissing=NULL, indLab=NULL,
auxiliary=NULL, ...)
```

Arguments

`model`	TBA
`data`	TBA
`package`	TBA
`simMissing`	TBA
`indLab`	TBA
`auxiliary`	TBA
`...`	TBA

Value

TBA

Author(s)

Patrick Miller (Univeristy of Notre Dame; <pmille13@nd.edu>) Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.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")

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 (University of Kansas; <schoemann@ku.edu>), Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

`SimResult` for the object input

Examples

```
loading1 <- matrix(0, 6, 1)
loading1[1:6, 1] <- NA
loading2 <- loading1
loading2[6,1] <- 0
LX1 <- bind(loading1, 0.7)
LX2 <- bind(loading2, 0.7)
RPH <- binds(diag(1))
```

```
RTD <- binds(diag(6))
CFA.Model1 <- model(LY = LX1, RPS = RPH, RTE = RTD, modelType="CFA")
CFA.Model2 <- model(LY = LX2, RPS = RPH, RTE = RTD, 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)

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)
```

`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 (Univeristy of Notre Dame; <pmille13@nd.edu>), Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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

---

centralMoment *Calculate central moments of a variable*

Description

Calculate central moments of a variable

Usage

```
centralMoment(x, ord, weight = NULL)
```

Arguments

x	A vector of a variable
ord	The order of the moment (the maximum is 4).
weight	A vector of weights of each case

Value

The central moment value

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
# This function is not public.

# centralMoment(1:5, 2)
```

clean — *Extract only converged replications in the result objects*

Description

Extract only the replications that are converegent in all supplied result objects (SimResult)

Usage

```
clean(...)
```

Arguments

...	The target result objects (SimResult)

Value

The cleaned result objects

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
# No example
```

cleanSimResult — *Extract only converged replications in the result object*

Description

Extract only the replications that are converegent in a result object (SimResult)

Usage

```
cleanSimResult(object, converged=NULL)
```

Arguments

object	The target result object (SimResult)
converged	The replications to be extracted. If NULL, the converged slot in the result object will be used

Value

The cleaned result object

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
# No example
```

`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., "LY1_1", "LY2_2"), or the name of a matrix (e.g. "PS"), if the name of a matrix is used power for all parameters in that matrix will be returned. 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 (University of Kansas; <schoemann@ku.edu>), Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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
LX <- bind(loading, 0.7)
RPH <- binds(diag(1))
RTD <- binds(diag(6))
CFA.Model <- model(LY = LX, RPS = RPH, RTE = RTD, modelType="CFA")
dat <- generate(CFA.Model, 50)
out <- analyze(CFA.Model, dat)

# We will use only 5 replications to save time.
# In reality, more replications are needed.

# Specify both sample size and percent missing completely at random

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

Cpow <- continuousPower(Output, contN = TRUE, contMCAR = TRUE)
Cpow

Cpow2 <- continuousPower(Output, contN = TRUE, contMCAR = TRUE, pred=list(N = 200, pmMCAR = 0.3))
Cpow2

## End(Not run)
```

---

createData *TBA*

Description

TBA

Usage

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

Arguments

paramSet	TBA
n	Sample size
indDist	TBA
sequential	TBA
facDist	TBA
errorDist	TBA
indLab	TBA
modelBoot	TBA
realData	TBA

Value

TBA

Author(s)

Patrick Miller (Univeristy of Notre Dame; <pmille13@nd.edu>), Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.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")

param <- draw(CFA.Model)

dat <- createData(param[[1]], n = 200)
# Get the first-group parameter from the param object
```

draw *TBA*

Description

TBA

Usage

```
draw(model, maxDraw=50, misfitBounds=NULL, averageNumMisspec=FALSE,
optMisfit = NULL, optDraws=50, misfitType="f0",misfitOut=FALSE)
```

Arguments

model	TBA
maxDraw	TBA
misfitBounds	TBA
averageNumMisspec	TBA
optMisfit	TBA
optDraws	TBA
misfitType	TBA
misfitOut	TBA

Value

TBA

Author(s)

Patrick Miller (Univeristy of Notre Dame; <pmille13@nd.edu>), Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.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")

param <- draw(CFA.Model)
```

extractLavaanFit *Extract fit indices from the lavaan object*

Description

Extract fit indices from the lavaan object

Usage

```
extractLavaanFit(Output)
```

Arguments

Output	The lavaan object

Value

The renamed vector of fit measures

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
# No example
```

find2Dhist *Fit the 2D Kernel Density Estimate*

Description

Fit the 2D Kernel Density Estimate to a pair of variables

Usage

```
find2Dhist(vec1, vec2)
```

Arguments

vec1	Variable 1
vec2	Variable 2

Value

The 2D Kernel Density Estimate based on each pair of values in vec1 and vec2

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
# No example
```

`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)
```

Arguments

`beta`	Regression coefficient matrix
`factorMean`	Total (model-implied) factor (indicator) means. The default is that all total factor means are 0.

Value

A vector of factor (indicator) intercepts

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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)
```

Arguments

`beta`	Regression coefficient matrix
`alpha`	Factor (indicator) intercept. The default is that all factor intercepts are 0.

Value

A vector of factor (indicator) total means

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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)
```

Arguments

beta	Regression coefficient matrix
corPsi	Factor or indicator residual correlations.
totalVarPsi	Factor or indicator total variances. The default is that all factor or indicator total variances are 1.

Value

A vector of factor (indicator) residual variances

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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)
```

Arguments

`beta`	Regression coefficient matrix
`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.

Value

A matrix of factor (model-implied) total covariance

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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)
```

Arguments

`beta`	Regression coefficient matrix
`corPsi`	Factor or indicator residual correlations.
`residualVarPsi`	Factor or indicator residual variances.

Value

A vector of factor (indicator) total variances

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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)
```

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.

Value

A vector of indicator (measurement) intercepts.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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)
```

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.

Value

A vector of indicator total means.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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)
```

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.

Value

A vector of indicator residual variances.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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)
```

Arguments

lambda	Factor loading matrix
totalFactorCov	Total (model-implied) covariance matrix among factors.
residualVarTheta	Indicator (measurement) residual variances.

Value

A vector of indicator total variances.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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)
```

---

findphist *Find the density (likelihood) of a pair value in 2D Kernel Density Estimate*

Description

Find the density (likelihood) of a pair value in 2D Kernel Density Estimate

Usage

```
findphist(value, hist)
```

Arguments

value	A target pair of values
hist	A 2D Binned Kernel Density Estimate

Value

The probability (density) of the target pair of value

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
# No example
```

`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 (University of Kansas; <psunthud@ku.edu>)

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 parmaeter 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 parmaeter 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.
- NaN 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 (University of Kansas; <psunthud@ku.edu>)

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
LX <- bind(loading, 0.4)
RPH <- binds(diag(1))
RTD <- binds(diag(6))
CFA.Model <- model(LY = LX, RPS = RPH, RTE = RTD, modelType="CFA")
```

```
# Specify both sample size and percent missing completely at random
Output <- sim(NULL, model=CFA.Model, n=seq(100, 200, 20), pmMCAR=c(0, 0.1, 0.2))
pow <- getPower(Output)
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 (University of Kansas; <psunthud@ku.edu>)

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)
```

findRowZero — *Find rows in a matrix that all elements are zero in non-fixed subset rows and columns.*

Description

Find rows in a matrix that all elements are zero in non-fixed subset rows and columns. This function will be used in the `findRecursiveSet` function

Usage

```
findRowZero(square.matrix, is.row.fixed = FALSE)
```

Arguments

`square.matrix`	Any square matrix
`is.row.fixed`	A logical vector with the length equal to the dimension of the `square.matrix`. If `TRUE`, the function will skip examining this row.

Value

A vector of positions that contain rows of all zeros

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
# No example
```

findTargetPower — *Find a value of varying parameters that provides a given value of power.*

Description

Find a value of varying parameters that provides a given value of power. This function can deal with only one varying parameter only (`findPower` can deal with more than one varying parameter).

Usage

```
findTargetPower(iv, dv, power)
```

Arguments

`iv`	A vector of the target varying parameter
`dv`	A `data.frame` of the power table of target parameters
`power`	A desired power.

Value

The value of the target varying parameter providing the desired power. If the value is NA, there is no value in the domain of varying parameters that provide the target power. If the value is the minimum value of the varying parameters, it means that the minimum value has already provided enough power. The value of varying parameters that provides exact desired power may be lower than the minimum value.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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.
- findPower to find a value of varying parameters that provides a given value of power, which can deal with more than one varying parameter

Examples

```
# No example
```

---

fitMeasuresChi *Find fit indices from the discrepancy values of the target model and null models.*

Description

Find fit indices from the discrepancy values of the target model and null models. This function is modified from the fitMeasures function in lavaan package

Usage

```
fitMeasuresChi(X2, df, p, X2.null, df.null, p.null, N, fit.measures="all")
```

Arguments

X2	The chi-square value of the target model
df	The degree of freedom of the target model
p	The p vlaue of the target model
X2.null	The chi-square value of the null model
df.null	The degree of freedom of the null model
p.null	The p value of the null model
N	Sample size
fit.measures	The list of selected fit measures

Value

A vector of fit measures

Author(s)

Yves Rosseel in the lavaan package Modified by Sunthud Pornprasertmanit (University of Kansas; psunthud@ku.edu)

Examples

```
# No example
```

---

generate *TBA*

Description

TBA

Usage

```
generate(model, n, maxDraw=50, misfitBounds=NULL, misfitType="f0",
averageNumMisspec=FALSE, optMisfit=NULL, optDraws=50,
indDist=NULL, sequential=FALSE,
facDist=NULL, errorDist=NULL, indLab=NULL, modelBoot=FALSE, realData=NULL,
params=FALSE)
```

Arguments

model	TBA
n	TBA
maxDraw	TBA
misfitBounds	TBA
misfitType	TBA
averageNumMisspec	TBA
optMisfit	TBA
optDraws	TBA
indDist	TBA
sequential	TBA
facDist	TBA
errorDist	TBA
indLab	TBA
modelBoot	TBA
realData	TBA
params	TBA

Value

TBA

Author(s)

Patrick Miller (Univeristy of Notre Dame; <pmille13@nd.edu>) Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.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")

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

---

getCondQtile *Get a quantile of a variable given values of predictors*

Description

Find a quantile of a variable. If the predictors are specified, the result will provide the conditional quantile given specified value of predictors. The quantreg package is used to find conditional quantile.

Usage

```
getCondQtile(y, x=NULL, xval=NULL, df = 0, qtile = 0.5)
```

Arguments

y	The variable that users wish to find a quantile from
x	The predictors variables. If NULL, the unconditional quantile of the y is provided.
xval	The vector of predictors' values that users wish to find the conditional quantile from. If "all" is specified, the function will provide the conditional quantile of every value in x.
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.
qtile	The quantile rank.

Value

A (conditional) quantile value

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- getCutoff for finding fit indices cutoffs using conditional quantiles

Examples

```
# No example
```

---

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

Description

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

Usage

```
getCutoff(object, alpha, revDirec = FALSE, usedFit = NULL, ...)
```

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.
...	Additional arguments.

Value

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

Methods

signature(object="data.frame") This method will find the fit indices cutoff given a specified alpha level. The additional arguments are predictor, predictorVal, and df, which allows the fit indices predicted by any arbitrary independent variables (such as sample size or percent MCAR). The predictor is the data.frame of the predictor values. The number of rows of the predictor argument should be equal to the number of rows in the object. The predictorVal is the values of predictor that researchers would like to find the fit indices cutoffs from. The df is 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.

signature(object="matrix") The details are similar to the method for data.frame

signature(object="SimResult") This method will find the fit indices cutoff given a specified alpha level. The additional arguments are `nVal`, `pmMCARval`, `pmMARval`, and `df`, which are needed when using varying sample sizes or percent missing across replications in `SimResult`. The `nVal` is the sample size value that researchers wish to find the fit indices cutoffs from. The `pmMCARval` is the percent missing completely at random value that researchers wish to find the fit indices cutoffs from. The `pmMARval` is the percent missing at random value that researchers wish to find the fit indices cutoffs from. The `df` is 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.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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
LX <- bind(loading, loadingValues)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- binds(latent.cor, 0.5)
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTD <- binds(error.cor)
CFA.Model <- model(LY = LX, RPS = RPH, RTE = RTD, 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)
getCutoff(Output, 0.05)

# Finding the cutoff when the sample size is varied.
Output2 <- sim(NULL, model=CFA.Model, n=seq(50, 100, 10))
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, pmM
```

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 (University of Kansas; <psunthud@ku.edu>)

See Also

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

Examples

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

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

loading.alt <- matrix(0, 6, 2)
loading.alt[1:3, 1] <- NA
loading.alt[4:6, 2] <- NA
LX.ALT <- bind(loading.alt, 0.7)
latent.cor.alt <- matrix(NA, 2, 2)
diag(latent.cor.alt) <- 1
RPH.ALT <- binds(latent.cor.alt, "runif(1, 0.7, 0.9)")
```

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

getCutoffNested(Output.NULL.NULL, Output.NULL.ALT)

## 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 (University of Kansas; <psunthud@ku.edu>)

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:
loading.A <- matrix(0, 8, 2)
loading.A[1:3, 1] <- NA
loading.A[4:8, 2] <- NA
LX.A <- bind(loading.A, 0.7)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- binds(latent.cor, "runif(1, 0.7, 0.9)")
RTD <- binds(diag(8))
CFA.Model.A <- model(LY = LX.A, RPS = RPH, RTE = RTD, modelType="CFA")

loading.B <- matrix(0, 8, 2)
loading.B[1:4, 1] <- NA
loading.B[5:8, 2] <- NA
LX.B <- bind(loading.B, 0.7)
CFA.Model.B <- model(LY = LX.B, RPS = RPH, RTE = RTD, 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)

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

## End(Not run)
```

---

getKeywords *List of all keywords used in the* simsem *package*

Description

List of all keywords used in the simsem package

Usage

```
getKeywords()
```

Value

A list of all keywords used in this package

- `usedFit` Fit indices used as the default for providing output
- `usedFitPop` Population fit indices used as the default for providing input
- `optMin` The method picking the minimum value of misfit across misspecification sets
- `optMax` The method picking the maximum value of misfit across misspecification sets
- `optNone` Not using the optimization method

Author(s)

Sunthud Pornprasertmanit (University of Kansas; psunthud@ku.edu)

Examples

```
# This function is not a public function.

# getKeywords()
```

`getPopulation` *Extract the data generation population model underlying an object*

Description

This function will extract the data generation population model underlying an object. The target object can be `linkS4class{SimResult}`.

Usage

```
getPopulation(object, ...)
```

Arguments

`object`	The target object that you wish to extract the data generation population model from, which can be `linkS4class{SimResult}`.
`...`	An additional argument. The details can be seen when this funciton is applied to the `linkS4class{SimDataOut}`.

Value

Depends on the class of `object`.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- SimResult for result object

Examples

```
# See each class for an example.
```

---

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 at least one randomly varying parameter (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., "LY1_1", "LY2_2"), or the name of a matrix (e.g. "PS"), if the name of a matrix is used power for all parameters in that matrix will be returned. 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 (University of Kansas; <schoemann@ku.edu>), Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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
LX <- bind(loading, 0.7)
RPH <- binds(diag(1))
RTD <- binds(diag(6))
CFA.Model <- model(LY = LX, RPS = RPH, RTE = RTD, modelType="CFA")

# We will use only 5 replications to save time.
# In reality, more replications are needed.

# Specify both sample size and percent missing completely at random
Output <- sim(NULL, model=CFA.Model, n=seq(100, 200, 20), pmMCAR=c(0, 0.1, 0.2))
summary(Output)

getPower(Output)

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, revDirec = FALSE, usedFit=NULL, ...)
```

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. This argument can be missing if the SimResult is specified in the altObject and the SimResult of the null model is specified.
revDirec	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	Vector of names of fit indices that researchers wish to getCutoffs from. The default is to getCutoffs of all fit indices.
...	Additional arguments

Value

List of power given different fit indices.

Methods

signature(altObject="data.frame", cutoff="vector") This method will find the fit indices indicated in the altObject that provides worse fit than the cutoff. The additional arguments are predictor, predictorVal, condCutoff, and df, which allows the fit indices predicted by any arbitrary independent variables (such as sample size or percent MCAR). The predictor is the data.frame of the predictor values. The number of rows of the predictor argument should be equal to the number of rows in the object. The predictorVal is the values of predictor that researchers would like to find the power from. The condCutoff is a logical. If TRUE, the cutoff is applicable only a given value of predictorVal. If FALSE, the cutoff is applicable in any values of predictor. The df is 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.

signature(altObject="matrix", cutoff="vector") The details are similar to the method for altObject="data.frame" and cutoff="vector".

signature(altObject="SimResult", cutoff="vector") This method will find the fit indices indicated in the altObject that provides worse fit than the cutoff. The additional arguments are nVal, pmMCARval, pmMARval, condCutoff, and df, which are needed when using varying sample sizes or percent missing across replications in SimResult. The nVal is the sample size value that researchers wish to find the fit indices cutoffs from. The pmMCARval is the percent missing completely at random value that researchers wish to find the fit indices cutoffs

from. The pmMARval is the percent missing at random value that researchers wish to find the fit indices cutoffs from. The condCutoff is a logical. If TRUE, the cutoff is applicable only a given set of nVal, pmMCARval, and pmMARval. If FALSE, the cutoff is applicable in any values of sample size and percent missing. The df is 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.

signature(altObject="SimResult", cutoff="missing") The details are similar to the method for altObject="SimResult" and cutoff="vector". The cutoff argument must not be specified. Rather, the nullObject, which is an additional argument of this method, is required. The nullObject is the SimResult that contains the simulation result from fitting the null model by the data from the null model.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

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

Examples

```
## Not run:
loading.null <- matrix(0, 6, 1)
loading.null[1:6, 1] <- NA
LX.NULL <- bind(loading.null, 0.7)
RPH.NULL <- binds(diag(1))
RTD <- binds(diag(6))
CFA.Model.NULL <- model(LY = LX.NULL, RPS = RPH.NULL, RTE = RTD, 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)
Cut.NULL <- getCutoff(Output.NULL, 0.95)

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

Output.ALT <- sim(5, n=500, model=CFA.Model.NULL, generate=CFA.Model.ALT)
getPowerFit(Output.ALT, cutoff=Cut.NULL)
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"))

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)
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, ...)
```

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.
...	Additional arguments

Value

List of power given different fit indices.

Methods

signature(altNested="SimResult", altParent="SimResult", cutoff="vector") This method will find the the differences in fit indices from altNested and altParent that provides worse fit than the cutoff. The additional arguments are revDirec, usedFit, nVal, pmMCARval, pmMARval, condCutoff, and df, which are needed when using varying sample sizes or percent missing across replications in SimResult. The revDirec is whether to reverse a direction. 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. The usedFit is the vector of names of fit indices that researchers wish to get power from. The default is to get the powers of all fit indices. The nVal is the sample size value that researchers wish to find the fit indices cutoffs from. The pmMCARval is the percent missing completely at random value that researchers wish to find the fit indices cutoffs from. The pmMARval is the percent missing at random value that researchers wish to find the fit indices cutoffs from. The condCutoff is a logical. If TRUE, the cutoff is applicable only a given set of nVal, pmMCARval, and pmMARval. If FALSE, the cutoff is applicable in any values of sample size and percent missing. The df is 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.

signature(altNested="SimResult", altParent="SimResult", cutoff="missing") The details are similar to the method for altNested="SimResult", altParent="SimResult", and cutoff="vector". The cutoff argument must not be specified. Rather, the nullNested and nullParent, which are additional arguments of this method, are required. The nullNested is the SimResult that saves the simulation result of the nested model when the nested model is TRUE. The nullParent is the SimResult that saves the simulation result of the parent model when the nested model is TRUE.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

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

Examples

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

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

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)

getPowerFitNested(Output.ALT.NULL, Output.ALT.ALT, nullNested=Output.NULL.NULL, nullParent=Output.NULL.ALT)
getPowerFitNested(Output.ALT.NULL, Output.ALT.ALT, cutoff=c(Chi=3.84, CFI=-0.10))

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)

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

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.
...	Additional arguments

Value

List of power given different fit indices.

Methods

signature(dat2Mod1="SimResult", dat2Mod2="SimResult", cutoff="vector") This method will find the the differences in fit indices from dat2Mod1 and dat2Mod2 that provides worse fit than the cutoff. The additional arguments are revDirec, usedFit, nVal, pmMCARval, pmMARval, condCutoff, and df, which are needed when using varying sample sizes or percent missing across replications in SimResult. The revDirec is whether to reverse a direction. The default is to count the proportion of the difference of fit indices that lower than the specified cutoffs, such as how many the difference in RMSEA in the alternative model that is lower than cutoffs. The direction can be reversed by setting as TRUE. The usedFit is the vector of names of fit indices that researchers wish to get power from. The default is to get the powers of all fit indices. The nVal is the sample size value that researchers wish to find the fit indices cutoffs from. The pmMCARval is the percent missing completely at random value that researchers wish to find the fit indices cutoffs from. The pmMARval is the percent missing at random value that researchers wish to find the fit indices cutoffs from. The condCutoff is a logical. If TRUE, the cutoff is applicable only a given set of nVal, pmMCARval, and pmMARval. If FALSE, the cutoff is applicable in any values of sample size and percent missing. The df is 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.

signature(dat2Mod1="SimResult", dat2Mod2="SimResult", cutoff="missing") The details are similar to the method for dat2Mod1="SimResult", dat2Mod2="SimResult", and cutoff="vector". The cutoff argument must not be specified. Rather, the dat1Mod1 and dat1Mod2, which are additional arguments of this method, are required. The dat1Mod1 is the SimResult that saves the simulation of analyzing Model 1 by datasets created from Model 1. The dat1Mod2 is the SimResult that saves the simulation of analyzing Model 2 by datasets created from Model 1. The another additional argument is onetailed that is to derive the cutoff by using one-tailed test if specified as TRUE.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

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

Examples

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

loading.B <- matrix(0, 8, 2)
loading.B[1:4, 1] <- NA
loading.B[5:8, 2] <- NA
LX.B <- bind(loading.B, 0.7)
CFA.Model.B <- model(LY = LX.B, RPS = RPH, RTE = RTD, 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)

getPowerFitNonNested(Output.B.A, Output.B.B, dat1Mod1=Output.A.A, dat1Mod2=Output.A.B)
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

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

Arguments

`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.
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 (University of Kansas; <schoemann@ku.edu>)

See Also

- SimMissing for the alternative way to save missing data feature for using in the sim function.
- runMI for imputing missing data by multiple imputation and analyze the imputed data.

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
imposeMissing(data,cov=c(1,2),pmMCAR=.1)


imposeMissing(data,nforms=3)
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)))
imposeMissing(datac,cov=c(20,21),nforms=3)
imposeMissing(data,twoMethod=c(19,.8))
imposeMissing(datac,cov=21,prAttr=.1,timePoints=5)
```

---

interpolate *Find the value of one vector relative to a value of another vector by interpolation*

Description

Find the value of the resulting vector that have the position similar to the value of the baseline vector. If the starting value in the baseline vector is in between two elements, the resulting value will be predicted by linear interpolation.

Usage

```
interpolate(baselineVec, val, resultVec=NULL)
```

Arguments

baselineVec	The target vector to be used as a baseline. The resulting vector can be attached as the element names of this vector.
val	The value relative to the baseline vector to be used for projecting the resulting value
resultVec	The vector that the resulting value will be used to base their result form

Value

The interpolated value from the resulting vector relative to the value in the baseline vector

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
# No Example
```

---

kStat *Calculate the* k*-statistic of a variable*

Description

Calculate the *k*-statistic (i.e., unbiased estimator of a cumulant) of a variable

Usage

```
kStat(x, ord)
```

Arguments

x	A vector of a variable
ord	The order of the *k*-statistics (The maximum value is 4).

Value

The k-statistic value

Author(s)

Sunthud Pornprasertmanit (University of Kansas; psunthud@ku.edu)

Examples

```
# This function is not a public function.

# kStat(1:5, 4)
```

kurtosis *Finding excessive kurtosis*

Description

Finding excessive kurtosis (g2) of an object

Usage

```
kurtosis(object, ...)
```

Arguments

object	An object used to find a excessive kurtosis, which can be a vector or a distribution object.
...	Other arguments such as the option for reversing the distribution.

Details

The excessive kurtosis computed is g2. See the Wolfram Mathworld for the excessive kurtosis detail.

Value

A value of an excessive kurtosis with a test statistic if the sample excessive kurtosis is computed.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
kurtosis(1:5)
```

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 (University of Kansas; <psunthud@ku.edu>)

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:
# It does not work now.

#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[2:3, 1] <- NA
#path.A[3, 2] <- NA
#param.A <- model(LY=bind(loading), BE=bind(path.A), modelType="SEM")

#model.A <- simModel(param.A, indLab=c(paste("x", 1:3, sep=""), paste("y", 1:8, sep="")))
#out.A <- run(model.A, PoliticalDemocracy)

#path.B <- matrix(0, 3, 3)
#path.B[1:2, 3] <- NA
#path.B[1, 2] <- NA
#param.B <- simParamSEM(LY=loading, BE=path.B)
```

```
#model.B <- simModel(param.B, indLab=c(paste("x", 1:3, sep=""), paste("y", 1:8, sep="")))
#out.B <- run(model.B, PoliticalDemocracy)

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

#output.A.A <- runFit(model.A, PoliticalDemocracy, 5, misspec=misspec)
#output.A.B <- runFit(model.A, PoliticalDemocracy, 5, misspec=misspec, analyzeModel=model.B)
#output.B.A <- runFit(model.B, PoliticalDemocracy, 5, misspec=misspec, analyzeModel=model.A)
#output.B.B <- runFit(model.B, PoliticalDemocracy, 5, misspec=misspec)

# The output may contain some warnings here. When the number of replications increases (e.g., 1000), the wa
#likRatioFit(out.A, out.B, output.A.A, output.A.B, output.B.A, output.B.B)

## End(Not run)
```

`loadingFromAlpha` *Find standardized factor loading from coefficient alpha*

Description

Find standardized factor loading from coefficient alpha assuming that all items have equal loadings.

Usage

```
loadingFromAlpha(alpha, ni)
```

Arguments

`alpha`	A desired coefficient alpha value.
`ni`	A desired number of items.

Value

`result`	The standardized factor loadings that make desired coefficient alpha with specified number of items.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
loadingFromAlpha(0.8, 4)
```

`miPool` *Function to pool imputed results*

Description

The function takes a list of imputed results (parameters and standard errors) and returns pooled parameter estimates and standard errors.

Usage

```
miPool(Result.l)
```

Arguments

`Result.l`	List of SimModelOut used for pooling based on Rubin's rule.

Details

All parameter estimates are pooled by Rubin's (1987) rule. The chi-square statistics are pooled by the procedure proposed by Li, Meng, Raghunathan, and Rubin (1991; Equations 2.1, 2.2, 2.16, and 2.17). The resulting value from the pooled chi-square is F-statistic. The F-statistics is multiplied by the numerator degree of feedom to provide the value equivalent to chi-square value. This chi-square value will be used to find other chi-squared related fit indices: RMSEA, CFI, and TLI. SRMR, AIC, and BIC are pooled by Rubin's (1987) rule. The function for pooling chi-squared statistics is adapted from Craig Enders' SAS code from "http://psychology.clas.asu.edu/files/CombiningLikelihoodRatioChi-SquareStatisticsFromaMIAnalysis.sas".

Value

MIpool returns a list with pooled estimates, standard errors, fit indices and fraction missing information

`Estimates`	Pooled parameter estimates.
`SE`	Pooled standard errors.
`FMI.1`	Fraction of missing information for each parameter.
`FMI.2`	Fraction of missing information for each parameter.

Author(s)

Alexander M. Schoemann (University of Kansas; <schoemann@ku.edu>) Mijke Rhemtulla (University of Kansas; <mijke@ku.edu>) Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

References

Rubin, D.B. (1987) *Multiple Imputation for Nonresponse in Surveys.* J. Wiley & Sons, New York.
Li, K. H., Meng, X. L., Raghunathan, T. E., & Rubin, D. B. (1991). Significance levels from repeated p-values with multiply-imputed data. *Statistica Sinica, 1,* 65-92.

See Also

- miPoolVector for pooling multiple imputation results that providing in matrix or vector formats
- miPoolChi for pooling multiple imputated chi-square statistics
- runMI for imputing missing values by multiple imputation and analyzing the imputed datasets.

Examples

```
# No Example
```

`miPoolChi` *Function to pool chi-square statistics from the result from multiple imputation*

Description

The function combines likelihood ratio chi-square statistics from an analysis of multiply imputed data sets using the method proposed by Li, Meng, Raghunathan, and Rubin (1991, p. 74).

Usage

```
miPoolChi(chis, df)
```

Arguments

`chis`	A vector of chi-square statistics
`df`	Degree of freedom that the chi-square statistics are based on

Details

The chi-square statistics are pooled by the procedure proposed by Li, Meng, Raghunathan, and Rubin (1991; Equations 2.1, 2.2, 2.16, and 2.17).

Value

The resulting value from the pooled chi-square is F-statistic. If the denominator degree of freedom is large, the F value multiplied by the numerator degree of freedom will approximate the chi-square statistics.

Author(s)

Craig Enders originally wrote this function in SAS, http://psychology.clas.asu.edu/files/CombiningLikelihoodRatioChi-SquareStatisticsFromaMIAnalysis.sas. Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>) modified the function to run in R.

References

Li, K. H., Meng, X. L., Raghunathan, T. E., & Rubin, D. B. (1991). Significance levels from repeated p-values with multiply-imputed data. *Statistica Sinica, 1,* 65-92.

See Also

- miPool for pooling analysis results (parameters and standard errors) and returns pooled parameter estimates, standard errors, and fit indices.
- miPoolVector for pooling multiple imputation results that providing in matrix or vector formats

Examples

```
miPoolChi(c(89.864, 81.116, 71.500, 49.022, 61.986, 64.422, 55.256, 57.890, 79.416, 63.944), 2)
```

`miPoolVector` *Function to pool imputed results that saved in a matrix format*

Description

The function takes parameter estimates and standard errors of each imputed result and returns pooled parameter estimates and standard errors.

Usage

```
miPoolVector(MI.param, MI.se, imps)
```

Arguments

`MI.param`	A matrix of parameter estimates that the row represents parameter estimates from different imputations and the column represents parameter estimates of different target parameters.
`MI.se`	A matrix of standard errors that the row represents standard errors from different imputations and the column represents the standard errors of different target parameters.
`imps`	The number of imputations

Details

Parameters and standard errors are combined using Rubin's Rules (Rubin, 1987).

Value

MIpool returns a list with pooled estimates, standard errors, fit indices and fraction missing information

`Estimates`	Pooled parameter estimates.
`SE`	Pooled standard errors.
`FMI.1`	Fraction of missing information for each parameter.
`FMI.2`	Fraction of missing information for each parameter.

Author(s)

Alexander M. Schoemann (University of Kansas; <schoemann@ku.edu>) Mijke Rhemtulla (University of Kansas; <mijke@ku.edu>)

References

Rubin, D.B. (1987) Multiple Imputation for Nonresponse in Surveys. J. Wiley & Sons, New York.

See Also

- runMI for imputing missing values by multiple imputation and analyzing the imputed datasets.
- miPool TBA

Examples

```
param <- matrix(c(0.7, 0.1, 0.5,
0.75, 0.12, 0.54,
0.66, 0.11, 0.56,
0.74, 0.09, 0.55), nrow=4, byrow=TRUE)
SE <- matrix(c(0.1, 0.01, 0.05,
0.11, 0.023, 0.055,
0.10, 0.005, 0.04,
0.14, 0.012, 0.039), nrow=4, byrow=TRUE)
nimps <- 4
miPoolVector(param, SE, nimps)
```

---

miss *TBA*

Description

TBA

Usage

```
miss(cov = 0, pmMCAR = 0, pmMAR = 0, nforms = 0, itemGroups = list(),
timePoints = 1, twoMethod = 0, prAttr = 0, package="default", 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.
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.
package	TBA
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.
...	TBA

Details

TBA

Value

TBA

Author(s)

Alexander M. Schoemann (University of Kansas; <schoemann@ku.edu>), Patrick Miller (Univeristy of Notre Dame; <pmille13@nd.edu>), Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

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

Examples

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

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

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

#Impose missing
#dat <- run(Missing, dat)

#Analyze data
#out <- run(SimModel, dat)
```

```
#summary(out)

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

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

Description

Creates a data generation and 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 `SimMatrix` or `SimVector` built using `bind`. 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 `SimMatrix` or `SimVector` to arguments, or by specifying the number of groups.

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, modelType, indLab=NULL, facLab=NULL, ngroups=1, smartStart=TRUE)
```

Arguments

LY	Factor loading matrix from endogenous factors to Y indicators (need to be `SimMatrix` object).
PS	Residual covariance matrix among endogenous factors (need to be `SimMatrix` object).
RPS	Residual correlation matrix among endogenous factors (need to be `SimMatrix` object).
TE	Measurement error covariance matrix among Y indicators (need to be `SimMatrix` object).
RTE	Measurement error correlation matrix among Y indicators (need to be `SimMatrix` object).
BE	Regression coefficient matrix among endogenous factors (need to be `SimMatrix` object).
VTE	Measurement error variance of indicators (need to be `SimVector` object).
VY	Total variance of indicators (need to be `SimVector` object). NOTE: Either measurement error variance or indicator variance is specified. Both cannot be simultaneously specified.
VPS	Residual variance of factors (need to be `SimVector` object).
VE	Total variance of of factors (need to be `SimVector` object). 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 `SimVector` object).

AL	Endogenous factor intercept (need to be SimVector object).
MY	Overall Y indicator means. (need to be SimVector object). NOTE: Either measurement intercept of indicator mean can be specified. Both cannot be specified simultaneously.
ME	Total mean of endogenous factors (need to be SimVector object). NOTE: Either endogenous factor intercept or total mean of endogenous factor is specified. Both cannot be simultaneously specified.
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 x1, x2, ... xx.
facLab	Character vector of factor labels. If left blank, automatic labels will be generated as y1, y2, ... yx
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.
smartStart	Defaults to "TRUE" - If population parameter values are numbers, use these as starting values.

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 linkS4class{SimMatrix} or linkS4class{SimVector}. The linkS4class{SimSem} object can be passed to the function generate to generate data.

If multiple group data is desired, the user can optionally either specify the number of groups argument, or pass a list of linkS4class{SimMatrix} or linkS4class{SimVector} to any of the matrix arguments. The length of this list will be the number of groups. If only one argument is a list, all other arguments will be automatically replicated to that length, parameters will be identified in the same way, have the same population parameter value/distribution, and have the same misspecification. If only ngroups is specified, all arguments will be replicated in this fashion. If equality constraints are present during the automatic replication, these parameters will be constrained to be equal across groups.

Value

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

Author(s)

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

See Also

- sim for simulations using the linkS4class{SimSem} template.
- generate To generate data using the linkS4class{SimSem} template.
- analyze To analyze real or generated data using the linkS4class{SimSem} template.
- draw To draw parameters using the linkS4class{SimSem} template.

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")
```

---

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 (University of Kansas; <psunthud@ku.edu>)

Examples

```
multipleAllEqual(1:5, 1:5, seq(2, 10, 2)/2)
multipleAllEqual(1:5, 1:6, seq(2, 10, 2)/2)
```

`overlapHist` *Plot overlapping histograms*

Description

Plot overlapping histograms

Usage

```
overlapHist(a, b, colors=c("red","blue","purple"), breaks=NULL, xlim=NULL,
ylim=NULL, main=NULL, xlab=NULL, swap=FALSE)
```

Arguments

`a`	Data for the first histogram
`b`	Data for the second histogram
`colors`	Colors for the first histogram, the second histogram, and the overlappling areas.
`breaks`	How many breaks users used in each histogram (should not be used)
`xlim`	The range of x-axis
`ylim`	The range of y-axis
`main`	The title of the figure
`xlab`	The labels of x-axis
`swap`	Specify `TRUE` to plot `b` first and then `a`. The default is `FALSE` to plot `a` first and then `b`.

Value

None. This function will plot only.

Author(s)

Chris Miller provided this code on http://chrisamiller.com/science/2010/07/20/transparent-overlapping-h
The code is modified by Sunthud Pornprasertmanit (University of Kansas; psunthud@ku.edu)

Examples

```
# This function is not a public function.

# a <- rnorm(10000, 0, 1)
# b <- rnorm(10000, 1, 1.5)
# overlapHist(a, b, main="Example")
```

`plot3DQtile` *Build a persepctive plot or contour plot of a quantile of predicted values*

Description

Build a persepctive plot or contour plot of a quantile of predicted values

Usage

```
plot3DQtile(x, y, z, df=0, qtile=0.5, useContour=TRUE, xlab=NULL,
ylab=NULL, zlab=NULL, main=NULL)
```

Arguments

`x`	The values of the first variable (e.g., a vector of sample size)
`y`	The values of the second variable (e.g., a vector of percent missing)
`z`	The values of the dependent variable
`df`	The degree of freedom in spline method
`qtile`	The quantile values used to plot a graph
`useContour`	If `TRUE`, use contour plot. If `FALSE`, use perspective plot.
`xlab`	The labels of x-axis
`ylab`	The labels of y-axis
`zlab`	The labels of z-axis
`main`	The title of the graph

Value

None. This function will plot only.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; psunthud@ku.edu)

Examples

```
# No example
```

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

Description

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

Usage

```
plotCutoff(object, ...)
```

Arguments

`object`	The object (`SimResult` or `data.frame`) that contains values of fit indices in each distribution.
`...`	Other arguments specific to different types of object you pass in the function.

Value

NONE. Only plot the fit indices distributions.

Details in ...

- `cutoff`: A priori cutoffs for fit indices, saved in a vector
- `cutoff2`: Another set of priori cutoffs for fit indices, saved in a vector
- `alpha`: A priori alpha level to getCutoffs of fit indices (do not specify when you have `cutoff`)
- `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 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.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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
LX <- bind(loading, loadingValues)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- binds(latent.cor, 0.5)
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTD <- binds(error.cor)
CFA.Model <- model(LY = LX, RPS = RPH, RTE = RTD, 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)
plotCutoff(Output, 0.05, usedFit=c("RMSEA", "SRMR", "CFI", "TLI"))

# Varying N
Output2 <- sim(NULL, n=seq(450, 500, 10), model=CFA.Model)
plotCutoff(Output2, 0.05)

# Varying N and pmMCAR
Output3 <- sim(NULL, n=seq(450, 500, 10), pmMCAR=c(0, 0.05, 0.1, 0.15), model=CFA.Model)
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 (University of Kansas; <psunthud@ku.edu>)

See Also

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

Examples

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

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

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 (University of Kansas; <psunthud@ku.edu>)

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:
loading.A <- matrix(0, 8, 2)
loading.A[1:3, 1] <- NA
loading.A[4:8, 2] <- NA
LX.A <- bind(loading.A, 0.7)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- binds(latent.cor, "runif(1, 0.7, 0.9)")
RTD <- binds(diag(8))
CFA.Model.A <- model(LY = LX.A, RPS = RPH, RTE = RTD, modelType="CFA")

loading.B <- matrix(0, 8, 2)
loading.B[1:4, 1] <- NA
loading.B[5:8, 2] <- NA
LX.B <- bind(loading.B, 0.7)
CFA.Model.B <- model(LY = LX.B, RPS = RPH, RTE = RTD, 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)

plotCutoffNonNested(Output.A.A, Output.A.B, Output.B.A, Output.B.B)
plotCutoffNonNested(Output.A.A, Output.A.B)
plotCutoffNonNested(Output.A.A, Output.A.B, onetailed=TRUE)

## End(Not run)
```

---

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

Description

Plot a distribution of a distribution object or data distribution object

Usage

```
plotDist(object, ...)
```

Arguments

object	The object to plot a distribution
...	Other arguments. Please see a class-specific method.

Value

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

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- SimDataDist for plotting a data distribution object
- VirtualDist for plotting a distribution object

Examples

```
gamma11 <- simGamma(1, 1)
plotDist(gamma11)

chi <- simChisq(5)
#dataDist <- simDataDist(chi, chi)
#plotDist(dataDist)
```

---

plotIndividualScatter *Plot an overlaying scatter plot visualizing the power of rejecting misspecified models*

Description

Plot the fit indices value against the value of predictors. The plot will include the fit indices value of the alternative models, the fit indices value of the null model (if specified), and the fit indices cutoffs (if specified).

Usage

```
plotIndividualScatter(altVec, nullVec=NULL, cutoff=NULL, x, main = NULL)
```

Arguments

altVec	The vector saving the fit index distribution when the hypothesized model is FALSE.
nullVec	The vector saving the fit index distribution when the hypothesized model is TRUE.
cutoff	A priori cutoff
x	The data.frame of the predictor values. The number of rows of the x argument should be equal to the number of rows in the object.
main	The title of the graph

Value

NONE. Only plot the fit indices distributions.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- plotPowerFit

Examples

```
# No example
```

plotLogisticFit *Plot multiple logistic curves for predicting whether rejecting a misspecified model*

Description

This function will find the fit indices cutoff values if not specified, then check whether the hypothesized model is rejected in each dataset, and plot the logistic curve given the value of predictors.

Usage

```
plotLogisticFit(altObject, nullObject=NULL, cutoff=NULL,
usedFit=NULL, x, xval, alpha=0.05, useContour=TRUE, df=0)
```

Arguments

altObject	The result object (data.frame) saves the simulation result of fitting the hypothesized model when the hypothesized model is FALSE.
nullObject	The result object (data.frame) 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
x	The data.frame of the predictor values. The number of rows of the x argument should be equal to the number of rows in the object.
xval	The values of predictor that researchers would like to find the fit indices cutoffs from.
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.
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

NONE. Only plot the fit indices distributions.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- plotPowerFit

Examples

```
# No example
```

plotMisfit *Plot the population misfit in parameter 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

Usage

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

Arguments

object	The result object, SimResult
usedFit	The fit indices used to plot. If the misParam is not specified, all fit indices are used. If the misParam is specified, the "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 (University of Kansas; <psunthud@ku.edu>)

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 5
ParamObject <- sim(20, n=500, Path.Model, misfitType="rmsea", paramOnly=TRUE)
#plotMisfit(ParamObject)

#plotMisfit(ParamObject, misParam=1:2)
```

`plotOverHist` *Plot multiple overlapping histograms*

Description

Plot multiple overlapping histograms and find the cutoff values if not specified

Usage

```
plotOverHist(altObject, nullObject, cutoff=NULL, usedFit=NULL, alpha=0.05,
cutoff2=NULL, cutoff3=NULL, cutoff4=NULL)
```

Arguments

`altObject`	The result object (`data.frame`) saves the simulation result of fitting the hypothesized model when the hypothesized model is `FALSE`.
`nullObject`	The result object (`data.frame`) saves the simulation result of fitting the hypothesized model when the hypothesized model is `TRUE`.
`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
`cutoff2`	Another vector of priori cutoffs for fit indices.
`cutoff3`	A vector of priori cutoffs for fit indices for the `altObject`.
`cutoff4`	Another vector of priori cutoffs for fit indices for the `altObject`.

Value

NONE. Only plot the fit indices distributions.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- `plotPowerFit`

Examples

```
# No example
```

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., "LY1_1", "LY2_2").
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 (University of Kansas; <psunthud@ku.edu>), Alexander M. Schoemann (University of Kansas; <schoemann@ku.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:
# Specify Sample Size by n
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- bind(loading, 0.4)
RPH <- binds(diag(1))
RTD <- binds(diag(6))
CFA.Model <- model(LY = LX, RPS = RPH, RTE = RTD, modelType="CFA")


# Specify both sample size and percent missing completely at random
Output <- sim(NULL, n=seq(100, 200, 20), pmMCAR=c(0, 0.1, 0.2), model=CFA.Model)
plotPower(Output, "1.LY1_1", contMCAR=FALSE)

## 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 (University of Kansas; <psunthud@ku.edu>)

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:
### Something is wrong here. Why the alternative data has the better fit than the null data.



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

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

datNull <- generate(CFA.Model.NULL, n=50, params=TRUE)
datAlt <- generate(CFA.Model.ALT, n=50, params=TRUE)
outNull <- analyze(CFA.Model.NULL, datNull)
outAlt <- analyze(CFA.Model.NULL, datAlt)
```

```
summaryFit(Output.NULL)
summaryFit(Output.ALT)

plotPowerFit(Output.ALT, nullObject=Output.NULL, alpha=0.05, usedFit=c("RMSEA", "CFI", "TLI", "SRMR"))
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"))

Output.NULL2 <- simResult(NULL, SimData.NULL, SimModel, n=seq(50, 250, 25))
Output.ALT2 <- simResult(NULL, SimData.ALT, SimModel, n=seq(50, 250, 25))

plotPowerFit(Output.ALT2, nullObject=Output.NULL2, alpha=0.05, usedFit=c("RMSEA", "CFI", "TLI", "SRMR"))
plotPowerFit(Output.ALT2, cutoff=Rule.of.thumb, alpha=0.05, usedFit=c("RMSEA", "CFI", "TLI", "SRMR"))

## End(Not run)
```

`plotPowerFitDf` *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. This function is similar to the `plotPowerFit` function but the input distributions are `data.frame`.

Usage

```
plotPowerFitDf(altObject, nullObject = NULL, cutoff = NULL, usedFit = NULL, alpha = 0.05, x = NU
```

Arguments

`altObject`	The result object (`data.frame`) saves the simulation result of fitting the hypothesized model when the hypothesized model is `FALSE`.
`nullObject`	The result object (`data.frame`) 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
`x`	The `data.frame` of the predictor values. The number of rows of the `x` argument should be equal to the number of rows in the `object`.
`xval`	The values of predictor that researchers would like to find the fit indices cutoffs from.
`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 (University of Kansas; <psunthud@ku.edu>)

See Also

- plotPowerFit

Examples

```
# No example
```

`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 (University of Kansas; <psunthud@ku.edu>)

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:
############# Still does not work. Check it back later

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

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

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)

plotPowerFitNested(Output.ALT.NULL, Output.ALT.ALT, nullNested=Output.NULL.NULL, nullParent=Output.NULL.AL
```

```
plotPowerFitNested(Output.ALT.NULL, Output.ALT.ALT, nullNested=Output.NULL.NULL, nullParent=Output.NULL.ALT

Output.NULL.NULL2 <- sim(NULL, n=seq(50, 500, 25), model=CFA.Model.NULL, generate=CFA.Model.NULL)
Output.ALT.NULL2 <- sim(NULL, n=seq(50, 500, 25), model=CFA.Model.NULL, generate=CFA.Model.ALT)
Output.NULL.ALT2 <- sim(NULL, n=seq(50, 500, 25), model=CFA.Model.ALT, generate=CFA.Model.NULL)
Output.ALT.ALT2 <- sim(NULL, n=seq(50, 500, 25), model=CFA.Model.ALT, generate=CFA.Model.ALT)

plotPowerFitNested(Output.ALT.NULL2, Output.ALT.ALT2, nullNested=Output.NULL.NULL2, nullParent=Output.NULL

plotPowerFitNested(Output.ALT.NULL2, Output.ALT.ALT2, nullNested=Output.NULL.NULL2, nullParent=Output.NULL

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 (University of Kansas; <psunthud@ku.edu>)

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:
# Still does not work. Check it later.

loading.A <- matrix(0, 8, 2)
loading.A[1:3, 1] <- NA
loading.A[4:8, 2] <- NA
LX.A <- bind(loading.A, 0.7)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- binds(latent.cor, "runif(1, 0.7, 0.9)")
RTD <- binds(diag(8))
CFA.Model.A <- model(LY = LX.A, RPS = RPH, RTE = RTD, modelType="CFA")

loading.B <- matrix(0, 8, 2)
loading.B[1:4, 1] <- NA
loading.B[5:8, 2] <- NA
LX.B <- bind(loading.B, 0.7)
CFA.Model.B <- model(LY = LX.B, RPS = RPH, RTE = RTD, 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)

plotPowerFitNonNested(Output.B.A, Output.B.B, dat1Mod1=Output.A.A, dat1Mod2=Output.A.B)
plotPowerFitNonNested(Output.B.A, Output.B.B, cutoff=c(AIC=0, BIC=0))

## End(Not run)
```

`plotPowerSig` *Plot multiple logistic curves given a significance result matrix*

Description

This function will plot the significance results given the value of predictors.

Usage

```
plotPowerSig(sig, x = NULL, xval=NULL, mainName = NULL, useContour = TRUE)
```

Arguments

`sig`	The `data.frame` of a significance result, which contains only `TRUE` for significance and `FALSE` for not significance.
`x`	The `data.frame` of the predictor values. The number of rows of the `x` argument should be equal to the number of rows in the `object`.
`xval`	The values of predictor that researchers would like to find the fit indices cutoffs from.
`mainName`	A vector of the titles of the graphs
`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 (University of Kansas; <psunthud@ku.edu>)

See Also

- `plotPower`

Examples

```
# No example
```

plotQtile *Build a scatterplot with overlaying line of quantiles of predicted values*

Description

Build a scatterplot with overlaying line of quantiles of predicted values

Usage

```
plotQtile(x, y, df=0, qtile=NULL, ...)
```

Arguments

x	The values of the independent variable (e.g., a vector of sample size)
y	The values of the dependent variable
df	The degree of freedom in spline method
qtile	The quantile values used to plot a graph
...	Other arguments in the plot command

Value

None. This function will plot only.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; psunthud@ku.edu)

Examples

```
# No example
```

plotScatter *Plot overlaying scatter plots visualizing the power of rejecting misspecified models*

Description

This function will find the fit indices cutoff values if not specified and then plot the fit indices value against the value of predictors. The plot will include the fit indices value of the alternative models, the fit indices value of the null model (if specified), and the fit indices cutoffs.

Usage

```
plotScatter(altObject, nullObject=NULL, cutoff=NULL, usedFit = NULL, x, alpha=0.05, df=5)
```

Arguments

altObject	The result object (data.frame) saves the simulation result of fitting the hypothesized model when the hypothesized model is FALSE.
nullObject	The result object (data.frame) 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.
x	The data.frame of the predictor values. The number of rows of the x argument should be equal to the number of rows in the object.
alpha	A priori alpha level
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

NONE. Only plot the fit indices distributions.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- plotPowerFit

Examples

```
# No example
```

---

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

Description

Find the discrepancy value between two means and covariance matrices

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

Details

The discrepancy value (F_0; Browne & Cudeck, 1992) is calculated by

$$F_0 = tr\left(\tilde{\Sigma}\Sigma^{-1}\right) - \log\left|\tilde{\Sigma}\Sigma^{-1}\right| - p + (\tilde{\mu} - \mu)' \Sigma^{-1} (\tilde{\mu} - \mu) .$$

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.

Value

The discrepancy between two means and covariance matrices

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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.

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")

Details

The root mean squared error of approximation (RMSEA) is calculated by

$$RMSEA = \sqrt{\frac{F_0}{df}}$$

where F_0 is the discrepancy value between two means vectors and covariance matrices (see popDiscrepancy) and df is the degree of freedom in the real model.

The standardized root mean squared residual can be calculated by

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

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

Value

The vector of the misfit indices

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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)
```

---

predProb *Function to get predicted probabilities from logistic regression*

Description

Function to get predicted probabilities from logistic regression

Usage

```
predProb(newdat, glmObj)
```

Arguments

newdat	A vector of values for all predictors, including the intercept
glmObj	An object from a fitted glm run with a logit link

Value

Predictive probability of success given the values in the newdat argument.

Author(s)

Alexander M. Schoemann (University of Kansas; <schoemann@ku.edu>)

See Also

- continuousPower
- getPower

Examples

```
# No example
```

---

printIfNotNull *Provide basic summary of each object if that object is not NULL.*

Description

Provide basic summary of each object if that object is not NULL. This function is mainly used in the summary function from the linkS4class{SimSet} object.

Usage

```
printIfNotNull(object, name=NULL)
```

Arguments

object	The target object to be printed, which can be linkS4class{SimMatrix}, linkS4class{SymMatrix}, or linkS4class{SimVector}.
name	The name of the target object

Value

None. This function will print only.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; psunthud@ku.edu)

Examples

```
# This function is not public

# AL <- simVector(rep(NA, 5), "0")
# printIfNotNull(AL, "Factor mean")
```

pValue *Find p-values (1 - percentile)*

Description

This function will provide *p* value from comparing number and vector or the analytic result to the observed data (in lavaan) and the simulation result (in SimResult).

Usage

```
pValue(target, dist, ...)
```

Arguments

target	A value, multiple values, or a model output 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.
...	Other values that will be explained specifically for each class

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. If the target is a critical value (e.g., fit index cutoff) and the dist is the sampling distribution underlying the alternative hypothesis, this function can provide a statistical power.

Value

Mostly, this function provides a vector of *p* values based on the comparison. If the target is a model output object and dist is a result object, 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.

Methods

signature(target="numeric", dist="vector") This method will find the *p* value (quantile rank) of the target value on the dist vector. The additional arguments are revDirec, x , xval, condCutoff, and df. The revDirec is a logical argument whether to reverse the direction of comparison. If TRUE, the proportion of the dist that is lower than target value is reported. If FALSE, the proportion of the dist that is higher than the target value is reported. The x is the data.frame of the predictor values. The number of rows of the x argument should be equal to the number of rows in the dist. The xval is the values of predictor that researchers would like to find the fit indices cutoffs from. The condCutoff is a logical argument. If TRUE, the cutoff is applicable only a given value of xval. If FALSE, the cutoff is applicable in any

values of predictor. The df is 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.

signature(target="numeric", dist="data.frame") This method will find the *p* value of each columns in the dist based on the value specified in the target. The additional arguments are revDirec, x , xval, df, and asLogical. The revDirec is a logical vector whether to reverse the direction of comparison. If TRUE, the proportion of the dist that is lower than target value is reported. If FALSE, the proportion of the dist that is higher than the target value is reported. The x is the data.frame of the predictor values. The number of rows of the x argument should be equal to the number of rows in the dist. The xval is the values of predictor that researchers would like to find the fit indices cutoffs from. The df is 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. The asLogical is to provide the result as the matrix of significance result (TRUE) or just the proportion of significance result (FALSE).

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- SimResult to run a simulation study

Examples

```
## Not run:
########## Make the pValue comparing between lavaan and SimResult work




# Compare number with a vector
pValue(0.5, rnorm(1000, 0, 1))

# Compare numbers with a data frame
pValue(c(0.5, 0.2), data.frame(rnorm(1000, 0, 1), runif(1000, 0, 1)))

# 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
#model <- simParamCFA(LY=loading)
#SimModel <- simModel(model, indLab=paste("x", 1:9, sep=""))
#u2 <- simUnif(-0.2, 0.2)
#loading.trivial <- matrix(NA, 9, 3)
#loading.trivial[is.na(loading)] <- 0
#LY.trivial <- simMatrix(loading.trivial, "u2")
#mis <- simMisspecCFA(LY = LY.trivial)
#out <- run(SimModel, HolzingerSwineford1939)
#Output2 <- runFit(out, HolzingerSwineford1939, 20, mis)
#pValue(out, Output2)

## End(Not run)
```

pValueCondCutoff *Find a p value when the target is conditional (valid) on a specific value of a predictor*

Description

Find a *p* value when the target is conditional (valid) on a specific value of a predictor. That is, the target value is applicable only a given value of a predictor.

Usage

```
pValueCondCutoff(target, dist, revDirec = FALSE, x = NULL, xval = NULL, df = 0)
```

Arguments

target	A target value used to find p values.
dist	The comparison distribution, which can be a vector of numbers, a data frame, or a result object.
revDirec	A logical argument whether to reverse the direction of comparison. If TRUE, the proportion of the dist that is lower than target value is reported. If FALSE, the proportion of the dist that is higher than the target value is reported.
x	the data.frame of the predictor values. The number of rows of the x argument should be equal to the number of rows in the dist
xval	the values of predictor that researchers would like 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

A vector of *p* values based on the comparison.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- pValue

Examples

```
# No example
```

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 (University of Kansas; <psunthud@ku.edu>)

See Also

- SimResult to run a simulation study

Examples

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

#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)
#linearModel <- simParamCFA(LY=LY, PS=PS, TY=TY, AL=AL, TE=TE)

#LY2 <- matrix(1, 4, 2)
#LY2[,2] <- c(0, NA, NA, 3)
#unconstrainModel <- simParamCFA(LY=LY2, PS=PS, TY=TY, AL=AL, TE=TE)

#nested <- simModel(linearModel, indLab=paste("t", 1:4, sep=""))
#parent <- simModel(unconstrainModel, indLab=paste("t", 1:4, sep=""))

#outNested <- run(nested, Demo.growth)
#outParent <- run(parent, Demo.growth)

#loadingMis <- matrix(0, 4, 2)
#loadingMis[2:3, 2] <- NA
#LYmis <- simMatrix(loadingMis, "runif(1, -0.1, 0.1)")
#linearMis <- simMisspecCFA(LY=LYmis)

#simNestedNested <- runFit(model=nested, data=Demo.growth, nRep=10, misspec=linearMis)
#simNestedParent <- runFit(model=nested, data=Demo.growth, nRep=10, misspec=linearMis, analyzeModel=parent)

#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 (University of Kansas; <psunthud@ku.edu>)

See Also

- SimResult to run a simulation study

Examples

```
## Not run:
#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[2:3, 1] <- NA
#path.A[3, 2] <- NA
#param.A <- simParamSEM(LY=loading, BE=path.A)

#model.A <- simModel(param.A, indLab=c(paste("x", 1:3, sep=""), paste("y", 1:8, sep="")))
#out.A <- run(model.A, PoliticalDemocracy)

#path.B <- matrix(0, 3, 3)
#path.B[1:2, 3] <- NA
#path.B[1, 2] <- NA
#param.B <- simParamSEM(LY=loading, BE=path.B)

#model.B <- simModel(param.B, indLab=c(paste("x", 1:3, sep=""), paste("y", 1:8, sep="")))
#out.B <- run(model.B, PoliticalDemocracy)

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

#output.A.A <- runFit(model.A, PoliticalDemocracy, 5, misspec=misspec)
#output.A.B <- runFit(model.A, PoliticalDemocracy, 5, misspec=misspec, analyzeModel=model.B)
#output.B.A <- runFit(model.B, PoliticalDemocracy, 5, misspec=misspec, analyzeModel=model.A)
#output.B.B <- runFit(model.B, PoliticalDemocracy, 5, misspec=misspec)

# The output may contain some warnings here. When the number of replications increases (e.g., 1000), the wa
#pValueNonNested(out.A, out.B, output.A.A, output.A.B, output.B.A, output.B.B)

## End(Not run)
```

---

pValueVariedCutoff *Find a p value when the cutoff is specified as a vector given the values of predictors*

Description

Find a *p* value when the cutoff is specified as a vector given the values of predictors.

Usage

```
pValueVariedCutoff(cutoff, obtainedValue, revDirec = FALSE, x = NULL, xval = NULL)
```

Arguments

cutoff	A vector of values used to find p values. Each value in the vector should be the target value conditional (applicable) to each value of the predictors (x) respectively.
obtainedValue	The comparison distribution, which can be a vector of numbers, a data frame, or a result object.
revDirec	A logical argument whether to reverse the direction of comparison. If TRUE, the proportion of the dist that is lower than target value is reported. If FALSE, the proportion of the dist that is higher than the target value is reported.
x	the data.frame of the predictor values. The number of rows of the x argument should be equal to the number of rows in the dist
xval	the values of predictor that researchers would like to find the fit indices cutoffs from.

Value

A vector of *p* values based on the comparison.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- pValue

Examples

```
# No example
```

---

revText *Reverse the proportion value by subtracting it from 1*

Description

Reverse the proportion value by subtracting it from 1. This function can reverse a value reported in text, such as from "> .98" to "< .02"

Usage

```
revText(val)
```

Arguments

val	The value to be reversed

Value

The reversed value or text

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
# This is a private function.

# revText(.96)
# revText("> .60")
```

---

run *Run a particular object in* simsem *package.*

Description

Run a particular object such as running any distribution objects to create number.

Usage

```
run(object, ...)
```

Arguments

object	'simsem' object
...	any additional arguments, listed below.

Value

object	depends on particular object

Methods

signature(object = "SimNorm") No additional arguments. The function will random draw a number from normal distribution object.

signature(object = "SimUnif") No additional arguments. The function will random draw a number from uniform distribution object.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

This is the list of classes that can use run method.

1. SimNorm
2. SimUnif

Examples

```
n02 <- simNorm(0, 0.2)
run(n02)
```

---

runMI *Multiply impute and analyze data using lavaan*

Description

This function takes data with missing observations, multiple imputes the data, runs a SEM using lavaan and combines the results using Rubin's rules.

Usage

```
runMI(data.mat, data.model, m, miPackage="amelia", silent = FALSE, opts)
```

Arguments

data.mat	Data frame with missing observations.
data.model	Specification of the model to be analyzed. data.model can be either a simModel object or lavaan syntax
m	Number of imputations wanted
miPackage	Package to be used for impuation. Currently runMI only uses amelia for imputation
silent	TRUE if users do not wish to print number of imputations while running the function.
opts	A list of additional arguments to be passed to amelia for imputation.

Value

runMI returns a list with pooled estimates, standard errors, fit indices and fraction missing information

coef	Pooled parameter estimates. The order of parameter estimates corresponds to the order reported by Lavaan
se	Pooled standard errors. The order of standard errors corresponds to the order reported by Lavaan
fit	Pooled fit indices. The order of fit indices corresponds to the order reported by Lavaan
FMI.1	Fraction of missing information for each parameter. The order of fraction missing corresponds to the order of parameters reported by Lavaan
FMI.2	Fraction of missing information for each parameter. The order of fraction missing corresponds to the order of parameters reported by Lavaan

Author(s)

Patrick Miller(University of Kansas; <patr1ckm@ku.edu>) Alexander M. Schoemann (University of Kansas; <schoemann@ku.edu>)

References

Rubin, D.B. (1987) Multiple Imputation for Nonresponse in Surveys. J. Wiley & Sons, New York.

See Also

- miPool for pooling results from multiple imputation.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(data.mat,data.model,imps) {
  #Impute missing data
  imputed.l<-imputeMissing(data.mat,imps)

  #Run models on each imputed data set
  #Does this give results from each dataset in the list?

  imputed.results<-result.object(imputed.l[[1]],sim.data.model,10)

  imputed.results <- lapply(imputed.l,result.object,data.model,1)
  comb.results<-MIpool(imputed.results,imps)

  return(comb.results)

  }
```

---

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 put (the same model as the analysis model), the additional features can be seen when we run a summary function on the target object, such as bias in parameter estimates or percentage coverage.

Usage

```
setPopulation(target, population, ...)
```

Arguments

target	The target object that you wish to set the data generation population model. The target argument is linkS4class{SimResult}.
population	The population parameters used to put
...	An additional argument, such as, when values are saved in a set of matrices, a set of matrices that indicates the position of free parameters is needed in the ....

Value

The target object that is changed the parameter.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- SimResult for result object

Examples

```
# See each class for an example.
```

`sim` *TBA*

Description

TBA

Usage

```
sim(nRep, model, n, generate = NULL, rawData = NULL,
miss = NULL, fun=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,
aux = NULL,
seed = 123321, silent = FALSE, multicore = FALSE, cluster = FALSE, numProc = NULL,
paramOnly = FALSE, dataOnly=FALSE, ...)
```

Arguments

nRep	TBA
model	TBA
n	TBA
generate	TBA
rawData	TBA
miss	TBA
fun	TBA
pmMCAR	TBA
pmMAR	TBA
facDist	TBA
indDist	TBA
errorDist	TBA

sequential	TBA
modelBoot	TBA
realData	TBA
maxDraw	TBA
misfitType	TBA
misfitBounds	TBA
averageNumMisspec	TBA
optMisfit	TBA
optDraws	TBA
aux	TBA
seed	TBA
silent	TBA
multicore	TBA
cluster	TBA
numProc	TBA
paramOnly	TBA
dataOnly	TBA
...	TBA

Value

TBA

Author(s)

Patrick Miller (Univeristy of Notre Dame; <pmille13@nd.edu>) Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.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")

Output <- sim(20, CFA.Model,n=200)
summary(Output)
```

simBeta *Create random beta distribution object*

Description

Create random beta distribution object. Random beta distribution object will save shape parameters and non-centrality parameter.

Usage

```
simBeta(shape1, shape2, ncp=0)
```

Arguments

shape1	The first shape parameter (alpha)
shape2	The second shape parameter (beta)
ncp	Non-centrality parameter

Value

SimBeta	Random Beta Distribution object (SimBeta) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- VirtualDist for all distribution objects.

Examples

```
b11 <- simBeta(1, 1)
run(b11)
```

simBinom *Create random binomial distribution object*

Description

Create random binomial distribution object. Random binomial distribution object will save the number of trials and the probability of successes parameters.

Usage

```
simBinom(size, prob)
```

Arguments

size	The number of trials
prob	The probability of successes

Value

SimBinom	Random Binomial Distribution object (SimBinom) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- VirtualDist for all distribution objects.

Examples

```
b55 <- simBinom(5, 0.5)
run(b55)
summary(b55)
```

---

simCauchy *Create random Cauchy distribution object*

Description

Create random Cauchy distribution object. Random Cauchy distribution object will save the location and the scale parameters.

Usage

```
simCauchy(location = 0, scale = 1)
```

Arguments

location	The location parameter
scale	The scale parameter

Value

SimCauchy	Random Cauchy Distribution object (SimCauchy) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- VirtualDist for all distribution objects.

Examples

```
    c02 <- simCauchy(0, 2)
    run(c02)
summary(c02)
```

`simChisq` *Create random chi-squared distribution object*

Description

Create random chi-squared distribution object. Random chi-squared distribution object will save the degree of freedom and the non-centrality parameters.

Usage

```
simChisq(df, ncp=0)
```

Arguments

`df`	The degree of freedom
`ncp`	The non-centrality parameter

Value

`SimChisq`	Random Chi-squared Distribution object (`SimChisq`) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- `VirtualDist` for all distribution objects.

Examples

```
    chi5 <- simChisq(5)
    run(chi5)
summary(chi5)
```

simDataDist *Create a data distribution object.*

Description

TBA

Usage

```
simDataDist(..., p=NULL, keepScale=TRUE, reverse=FALSE)
```

Arguments

...	List of distribution objects. See VirtualDist for a list of possible distributions.
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.

Value

SimDataDist that saves analysis result from simulate data.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- SimResult for the type of resulting object

Examples

```
# Need an example
```

SimDataDist-class *Class* "SimDataDist"

Description

This class will provide the distribution of a dataset.

Objects from the Class

Objects can be created by simDataDist function. It can also be called from the form new("SimDataDist", ...).

Slots

p: Number of variables

dist: The list of marginal distribution objects, which will be used in a normal copula.

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).

Methods

- summaryTo summarize the object
- runTo create data from an object. There are three additional required objects: n = sample size, m = mean of variables, cm = covariance matrix of variables.
- plotDistTo plot a density distribution (for one variable) or a contour plot (for two variables). If the object has more than two variables, the var argument can be used to select the index of plotting variables. For two variables, the default is to have correlation of 0. To change a correlation, the r argument can be used. The xlim and ylim can be specified to set the ranges of variables.
- extractExtract elements from an object. The next argument is the position of the object to be extracted.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- simDataDist The constructor of this class.

Examples

```
# Need to be fixed

showClass("SimDataDist")

chisq3 <- simChisq(3)
chisq8 <- simChisq(8)
dist <- simDataDist(chisq3, chisq8)
```

```
m <- c(0, 0)
cm <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
n <- 20
# dat <- run(dist, n, m, cm)

plotDist(dist, r=0.2)
```

`simExp` *Create random exponential distribution object*

Description

Create random exponential distribution object. Random exponential distribution object will save the rate parameters.

Usage

```
simExp(rate = 1)
```

Arguments

`rate`	The rate parameter

Value

`SimExp`	Random Exponential Distribution object (`SimExp`) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- `VirtualDist` for all distribution objects.

Examples

```
exp2 <- simExp(2)
run(exp2)
summary(exp2)
```

simF *Create random F distribution object*

Description

Create random F distribution object. Random F distribution object will save the numerator and denominator degrees of freedom and the non-centrality parameters.

Usage

```
simF(df1, df2, ncp = 0)
```

Arguments

df1	The numerator degree of freedom
df2	The denominator degree of freedom
ncp	The non-centrality parameter

Value

SimF	Random F Distribution object (SimF) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- VirtualDist for all distribution objects.

Examples

```
f27 <- simF(2, 7)
run(f27)
summary(f27)
```

simFunction *Create function object*

Description

This function is a constrctor of a function object which can be used for data transformation. The aim of the object is to create a function but will use later in a simulation study. For example, set up a mean centering for a dataset for using in a simulation.

Usage

```
simFunction(fun, ...)
```

Arguments

fun	The desired function that will be used for data transformation
...	Additional arguments of the desired function.

Value

SimFunction that saves the function to use later in the simulation (within sim)

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- SimResult for how to use the simFunction in a simulation study

Examples

```
# The example still does not work
loading <- matrix(0, 9, 3)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loading[7:9, 3] <- NA
loading.start <- matrix("", 9, 3)
loading.start[1:3, 1] <- 0.7
loading.start[4:6, 2] <- 0.7
loading.start[7:9, 3] <- "runif(1, 0.6, 0.8)"

loading.trivial <- matrix("runif(1, -0.2, 0.2)", 9, 3)
loading.trivial[is.na(loading)] <- 0

LY <- bind(loading, loading.start, misspec=loading.trivial)

error.cor.trivial <- matrix("rnorm(1, 0, 0.1)", 9, 9)
diag(error.cor.trivial) <- 0

RTE <- binds(diag(9), misspec=error.cor.trivial)

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)

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

#loading <- matrix(0, 12, 4)
#loading[1:3, 1] <- NA
#loading[4:6, 2] <- NA
#loading[7:9, 4] <- NA
#loading[10:12, 3] <- NA
```

```
#path <- matrix(0, 4, 4)
#path[4, 1:3] <- NA

#analysis <- simParamSEM(BE=path, LY=loading)

#Model <- simModel(analysis)

# Find the products of indicators
#newFUN <- function(data, var1, var2, namesProd) {
# prod <- data[,var1] * data[,var2]
# colnames(prod) <- namesProd
# return(data.frame(data, prod))
#}

#fun <- simFunction(newFUN, var1=paste("y", 1:3, sep=""), var2=paste("y", 4:6, sep=""), namesProd=paste("y

# Real simulation will need more than just 10 replications
#Output <- simResult(10, Data.Mis, Model, objFunction=fun)
#summary(Output)
```

SimFunction-class *Class* "SimFunction"

Description

This class will save a function using for data transformation later in a simulation study within sim.

Objects from the Class

Objects can be created by simFunction. It can also be called from the form new("SimFunction", ...).

Slots

fun: The desired function that will be used for data transformation.

attribute: Additional arguments of the desired function.

callfun: The command that users used to create the object.

Methods

- summary To summarize the object
- run To use the object for data transformation.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- SimResult for how to use the simFunction in a simulation study

Examples

```
# The example still does not work

#showClass("SimFunction")

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

#loading <- matrix(0, 9, 3)
#loading[1:3, 1] <- NA
#loading[4:6, 2] <- NA
#loading[7:9, 3] <- NA
#loading.start <- matrix("", 9, 3)
#loading.start[1:3, 1] <- 0.7
#loading.start[4:6, 2] <- 0.7
#loading.start[7:9, 3] <- "u68"
#LY <- simMatrix(loading, loading.start)

#RTE <- symMatrix(diag(9))

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

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

#datGen <- simSetSEM(BE=BE, LY=LY, RPS=RPS, RTE=RTE)

#loading.trivial <- matrix(NA, 9, 3)
#loading.trivial[is.na(loading)] <- 0
#LY.trivial <- simMatrix(loading.trivial, "u2")

#error.cor.trivial <- matrix(NA, 9, 9)
#diag(error.cor.trivial) <- 0
#RTE.trivial <- symMatrix(error.cor.trivial, "n1")

#misGen <- simMisspecSEM(LY = LY.trivial, RTE = RTE.trivial)

#Data.Mis <- simData(datGen, 300, misspec=misGen)

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

#path <- matrix(0, 4, 4)
#path[4, 1:3] <- NA
```

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

#Model <- simModel(analysis)

# Find the products of indicators
#newFUN <- function(data, var1, var2, namesProd) {
# prod <- data[,var1] * data[,var2]
# colnames(prod) <- namesProd
# return(data.frame(data, prod))
#}

#fun <- simFunction(newFUN, var1=paste("y", 1:3, sep=""), var2=paste("y", 4:6, sep=""), namesProd=paste("y

# Real simulation will need more than just 10 replications
#Output <- simResult(10, Data.Mis, Model, objFunction=fun)
#summary(Output)

# Example of using the simfunction
#mc <- simFunction(newFUN, var1=1:3, var2=4:6, namesProd=paste("y", 10:12, sep=""))
#run(mc, attitude[,-1])
#summary(mc)
```

`simGamma` *Create random gamma distribution object*

Description

Create random gamma distribution object. Random gamma distribution object will save the shape and rate parameters.

Usage

```
simGamma(shape, rate = 1)
```

Arguments

`shape`	The shape parameter (alpha)
`rate`	The rate parameter (beta)

Value

`SimGamma`	Random Gamma Distribution object (`SimGamma`) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- `VirtualDist` for all distribution objects.

Examples

```
    g11 <- simGamma(1, 1)
    run(g11)
summary(g11)
```

`simGeom` *Create random geometric distribution object*

Description

Create random geometric distribution object. Random geometric distribution object will save the probability of successes parameters.

Usage

```
simGeom(prob)
```

Arguments

`prob`	The probability of successes

Value

`SimGeom`	Random Geometric Distribution object (`SimGeom`) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- `VirtualDist` for all distribution objects.

Examples

```
    geom5 <- simGeom(0.05)
    run(geom5)
summary(geom5)
```

simHyper *Create random hypergeometric distribution object*

Description

Create random hypergeometric distribution object. Random hypergeometric distribution object will save the numbers of successes, failures, and drawns parameters.

Usage

```
simHyper(m, n, k)
```

Arguments

m	The number of successes
n	The number of failures
k	The number of drawns

Value

SimHyper	Random Hypergeometric Distribution object (SimHyper) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- VirtualDist for all distribution objects.

Examples

```
hyp <- simHyper(20, 5, 10)
run(hyp)
summary(hyp)
```

simLnorm *Create random log normal distribution object*

Description

Create random log normal distribution object. Random log normal distribution object will save the mean and standard deviation (in log scale) parameters.

Usage

```
simLnorm(meanlog = 0, sdlog = 1)
```

Arguments

meanlog	The mean in log scale
sdlog	The standard deviation in log scale

Value

SimLnorm	Random Log Normal Distribution object (SimLnorm) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- VirtualDist for all distribution objects.

Examples

```
lognorm <- simLnorm(0, exp(1))
run(lognorm)
summary(lognorm)
```

---

simLogis *Create random logistic distribution object*

Description

Create random logistic distribution object. Random logistic distribution object will save the location and scale parameters.

Usage

```
simLogis(location = 0, scale = 1)
```

Arguments

location	The location parameter
scale	The scale parameter

Value

SimLogis	Random Logistic Distribution object (SimLogis) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- VirtualDist for all distribution objects.

Examples

```
    logis <- simLogis(0, 1)
    run(logis)
summary(logis)
```

`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, and starting values. This object can be represented factor loading matrix or regreesion coefficient matrix.

Objects from the Class

This object is created by `bind` function. Objects can be also created by calls of the form `new("SimMatrix", ...)`.

Slots

`free`: TBA

`popParam`: TBA

`misspec`: TBA

`symmetric`: TBA

Methods

`run` Draws starting values from the `"labels"` slot and show as a matrix sample.

`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 (University of Kansas; <psunthud@ku.edu>)

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
LX <- bind(loading, loadingValues)
summary(LX)
```

```
# run(LX)

LY <- bind(loading, "rnorm(1, 0.6, 0.05)")
summary(LY)
# run(LY)
```

SimMissing-class *Class* "SimMissing"

Description

Missing information imposing on the complete dataset

Objects from the Class

Objects can be created by simMissing function. It can also be called from the form new("SimMissing", ...).

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.

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.

package: The package used in multiple imputation. If "default", the full-information maximum likelihood is used.

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
- run To impose missing information into data

Author(s)

Patrick Miller(University of Kansas; <patr1ckm@ku.edu>) Alexander M. Schoemann (University of Kansas; <schoemann@ku.edu>) Kyle Lang (University of Kansas; <kylelang@ku.edu>) Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- imposeMissing for directly imposing missingness into a dataset.

Examples

```
# No Example
```

---

simNbinom *Create random negative binomial distribution object*

Description

Create random negative binomial distribution object. Random negative binomial distribution object will save the target number of successful trials and the probability of successes parameters.

Usage

```
simNbinom(size, prob)
```

Arguments

size	The target number of successful trials
prob	The probability of successes

Value

SimNbinom	Random Negative Binomial Distribution object (SimNbinom) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- VirtualDist for all distribution objects.

Examples

```
    nbinom <- simNbinom(5, 0.25)
    run(nbinom)
summary(nbinom)
```

simNorm *Create random normal distribution object*

Description

Create random normal distribution object. Random normal distribution object will save mean and standard deviation parameter.

Usage

```
simNorm(mean, sd)
```

Arguments

mean	Desired population mean
sd	Desired population standard deviation

Value

SimNorm	Random Normal Distribution object (SimNorm) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- VirtualDist for all distribution objects.

Examples

```
n02 <- simNorm(0, 0.2)
run(n02)
summary(n02)
```

simPois *Create random Poisson distribution object*

Description

Create random Poisson distribution object. Random Poisson distribution object will save the lambda parameters.

Usage

```
simPois(lambda)
```

Arguments

lambda	The lambda parameter (equal to the expected value of mean and variance)

Value

`SimPois` Random Poisson Distribution object (`SimPois`) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- `VirtualDist` for all distribution objects.

Examples

```
    pois5 <- simPois(5)
    run(pois5)
summary(pois5)
```

SimResult-class *Class* "SimResult"

Description

This class will save data analysis results from multiple replications and ready to find some useful statistics, such as fit indices cutoffs or power.

Objects from the Class

Objects can be created by `sim`. It can also be called from the form `new("SimResult", ...)`.

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`: Number of convergence replications

`seed`: Seed number.

`paramValue`: Population model underlying each simulated dataset.

`FMI1`: Fraction Missing Method 1.

`FMI2`: Fraction Missing Method 2.

`stdCoef`: Standardized coefficients from each replication

`n`: Sample size of the analyzed data.

`pmMCAR`: Percent missing completely at random.

`pmMAR`: Percent missing at random.

`timing`: Time elapsed in each phase of the simulation.

Methods

- getCutoff to getCutoff of fit indices based on a priori alpha level.
- getPowerFit to getPowerFit of rejection when the simResult is the alternative hypothesis and users specify cutoffs of the fit indices.
- plotCutoff to plot null hypothesis sampling distributions of fit indices with an option to draw fit indices cutoffs by specifying a priori alpha level.
- plotPowerFit to plot alternative hypothesis (and null hypothesis) with a priori cutoffs or alpha level.
- summary to summarize the result output
- summaryParam to summarize all parameter estimates
- anova 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. See anova for further details.
- summaryPopulation to summarize the data generation population underlying the simulation study.
- getPopulation to extract the data generation population underlying the simulation study. This method will return a data frame of the population underlying each replication.
- setPopulation to put the appropriate data generation model into the result object. If the appropriate data generation model is put (the same model as the analysis model), the bias in parameter estimates and standard errors will be able to be calculated by the summary function. The first argument is the result object. The second argument can be either data.frame of the population or SimSet of the population. See the 'modeling with covariate' in the manual for an example.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- sim for the constructor of this class

Examples

```
showClass("SimResult")
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- bind(loading, 0.7)
RPH <- binds(diag(1))
RTD <- binds(diag(6))
CFA.Model <- model(LY = LX, RPS = RPH, RTE = RTD, 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(Output)
getCutoff(Output, 0.05)
summaryParam(Output)
summaryPopulation(Output)
param <- getPopulation(Output)
Output <- setPopulation(Output, param)
```

SimSem-class *Class* "SimSem"

Description

TBA

Objects from the Class

TBA

Slots

pt: TBA

dgen: TBA

modelType: TBA

Methods

summary Get the summary of model specification

Author(s)

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

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
LX <- bind(loading, loadingValues)
summary(LX)

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

# Error Correlation Object
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTD <- binds(error.cor)

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

```
summary(CFA.Model)
#run(CFA.Model)

#CFA.Model2 <- extract(CFA.Model, y=1:3, e=1)
#summary(CFA.Model2)
```

`simT` *Create random t distribution object*

Description

Create random t distribution object. Random t distribution object will save the degree of freedom and the non-centrality parameters.

Usage

```
simT(df, ncp = 0)
```

Arguments

`df`	The degree of freedom
`ncp`	The non-centrality parameter

Value

`SimT`	Random t Distribution object (`SimT`) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- `VirtualDist` for all distribution objects.

Examples

```
    nct82 <- simT(8, ncp=2)
    run(nct82)
summary(nct82)
```

simUnif *Create random uniform distribution object*

Description

Create random uniform distribution object. Random uniform distribution object will save mean and standard deviation parameter.

Usage

```
simUnif(min, max)
```

Arguments

min	Lower bound of the distribution
max	Upper bound of the distribution

Value

SimUnif	Random Uniform Distribution object (SimUnif) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- VirtualDist for all distribution objects.

Examples

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

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, and starting values. This object can be represented mean, intercept, or variance vectors.

Objects from the Class

This object is created by bind function. Objects can be created by calls of the form new("SimVector", ...).

Slots

`free`: TBA

`popParam`: TBA

`misspec`: TBA

Methods

`run` Draws starting values from the `"labels"` slot and show as a vector sample.

`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 (University of Kansas; <psunthud@ku.edu>)

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)
#run(AL)
summary(AL)
summaryShort(AL)
```

---

simWeibull *Create random Weibull distribution object*

Description

Create random Weibull distribution object. Random Weibull distribution object will save the shape and scale parameters.

Usage

```
simWeibull(shape, scale = 1)
```

Arguments

`shape`	The shape parameter
`scale`	The scale parameter

Value

`SimWeibull`	Random Weibull Distribution object (`SimWeibull`) that save the specified parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- VirtualDist for all distribution objects.

Examples

```
    exWeibull <- simWeibull(2, 100)
    run(exWeibull)
summary(exWeibull)
```

---

skew *Find skewness*

Description

Finding skewness (g1) of an object

Usage

```
skew(object, ...)
```

Arguments

object	An object used to find a skewness, which can be a vector or a distribution object.
...	Other arguments such as the option for reversing the distribution.

Details

The skewness computed is g1. See the Wolfram Mathworld for the skewness detail.

Value

A value of a skewness with a test statistic if the sample skewness is computed.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
skew(1:5)
```

sortList *Sort two objects in a list*

Description

Sort two objects in a list by swapping the values of both objects so that the first object contains the lower value and the second object contains the larger value

Usage

```
sortList(object)
```

Arguments

object	The list with two objects (e.g., vector, matrix)

Value

The sorted list

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
# No example
```

subtractObject *Make a subtraction of each element in an object*

Description

Make a subtraction of each element in an object. For example, subtract the parameter estimates by the paramter values

Usage

```
subtractObject(object1, object2, ...)
```

Arguments

object1	The first object
object2	The second object
...	Additional arguments specific to each class

Value

The object after subtraction

Methods

signature(object1="SimRSet", object2="SimRSet") This function will find the bias by subtracting for parameter estimates from the real parameters.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
# This function is not public

#u89 <- simUnif(0.8, 0.9)
#loading <- matrix(0, 6, 2)
#loading[1:3, 1] <- NA
#loading[4:6, 2] <- NA
#loadingValues <- matrix(0, 6, 2)
#LX <- simMatrix(loading, "u89")
#startingValues(LX, 10)

#u89 <- simUnif(0.8, 0.9)
#loading <- matrix(0, 6, 2)
#loading[1:3, 1] <- NA
#loading[4:6, 2] <- NA
#loadingValues <- matrix(0, 6, 2)
#LX <- simMatrix(loading, "u89")
#latent.cor <- matrix(NA, 2, 2)
#diag(latent.cor) <- 1
#PH <- symMatrix(latent.cor, 0.5)
#error.cor <- matrix(0, 6, 6)
#diag(error.cor) <- 1
#TD <- symMatrix(error.cor)
#CFA.Model <- simSetCFA(LX = LX, PH = PH, TD = TD)
#result <- startingValues(CFA.Model, 10)
#summary(result)
```

---

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,...)
```

Arguments

object	SimResult to be summarized
...	any additional arguments, such as for the function with result object, digits argument is available to adjust digits in results, alpha is available to select a specific alpha for fit index cutoffs.

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.

See details in `popDiscrepancy` and `popMisfitMACS`

Author(s)

Alexander M. Schoemann (University of Kansas; <schoemann@ku.edu>) Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

`SimResult` for the result object input

Examples

```
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- bind(loading, 0.7)
RPH <- binds(diag(1))
RTD <- binds(diag(6))
CFA.Model <- model(LY = LX, RPS = RPH, RTE = RTD, 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)
summaryFit(Output)
```

`summaryMisspec` *Provide summary of model misspecification imposed across replications*

Description

This function will the amount of population misfit imposed in the real parameter

Usage

```
summaryMisspec(object,...)
```

Arguments

object	TBA
...	Additional arguments

Value

A data frame that provides the summary of model misspecification imposed on the real parameters

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

SimResult for the object input

Examples

```
# Incomplete

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 5
# ParamObject <- simResultParam(5, Path.Model, Path.Mis.Model)

# summaryMisspec(ParamObject)
```

---

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, ...)
```

Arguments

object	SimResult object being described
...	any additional arguments, such as for the function with result object, detail argument is available. If TRUE, it provides relative bias, standardized bias, and relative bias in standard errors.

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
- Average.Param: Parameter values or average values of parameters if random parameters are specified
- SD.Param: Standard Deviations of parameters. Appeared 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. Appeared 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

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

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(4)*, 330.
- Hoogland, J. J., & Boomsma, A. (1998). Robustness studies in covariance structure modeling. *Sociological Methods & Research, 26(3)*, 329.

See Also

SimResult for the object input

Examples

```
showClass("SimResult")
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- bind(loading, 0.7)
RPH <- binds(diag(1))
RTD <- binds(diag(6))
CFA.Model <- model(LY = LX, RPS = RPH, RTE = RTD, 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)
summaryParam(Output)
summaryParam(Output, detail=TRUE)
```

---

summaryPopulation *Summarize the data generation population model underlying an object*

Description

This function will summarize the data generation population model underlying an object. The target object can be linkS4class{SimResult}.

Usage

```
summaryPopulation(object)
```

Arguments

object	The target object that you wish to extract the data generation population model from, which can be linkS4class{SimResult}.

Value

None except using for linkS4class{SimResult} which the return value is a data.frame of the summary of population model across replications.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

- SimResult for result object

Examples

```
# See each class for an example.
```

---

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 (University of Kansas; <psunthud@ku.edu>)

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)
LX <- bind(loading, "runif(1, 0.8, 0.9)")
summaryShort(LX)
```

toFunction *Export the distribution object to a function command in text that can be evaluated directly.*

Description

Export the distribution object to a function command in text that can be evaluated directly.

Usage

```
toFunction(x)
```

Arguments

x	The distribution object used to be transformed

Value

The expression that is ready to be evaluated.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; psunthud@ku.edu)

See Also

VirtualDist for the distribution object that can be transformed to a function

Examples

```
u2 <- simUnif(-0.2, 0.2)
toFunction(u2)
```

twoTailedPValue *Find two-tailed* p *value from one-tailed* p *value*

Description

Find two-tailed *p* value from one-tailed *p* value

Usage

```
twoTailedPValue(vec)
```

Arguments

vec	A vector of one-tailed *p* value.

Value

A vector of two-tailed *p* value.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
# No example
```

---

validateCovariance *Validate whether all elements provides a good covariance matrix*

Description

Validate whether all elements provides a good covariance matrix

Usage

```
validateCovariance(resVar, correlation, totalVar = NULL)
```

Arguments

resVar	A vector of residual variances
correlation	A correlation matrix
totalVar	A vector of total variances

Value

Return TRUE if the covariance matrix is good

Author(s)

Sunthud Pornprasertmanit (University of Kansas; psunthud@ku.edu)

Examples

```
# No example
```

---

validateObject *Validate whether the drawn parameters are good.*

Description

Validate whether the drawn parameters are good (providing an identified model).

Usage

```
validateObject(paramSet)
```

Arguments

paramSet	A target set of parameters

Value

Return TRUE if the target parameters are good.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; psunthud@ku.edu)

Examples

```
# No example
```

---

validatePath *Validate whether the regression coefficient (or loading) matrix is good*

Description

Validate whether the regression coefficient (or loading) matrix is good

Usage

```
validatePath(path, var.iv, var.dv)
```

Arguments

path	A regression coefficient or loading matrix
var.iv	The variances of variables corresponding to the columns
var.dv	The variances of variables corresponding to the rows

Value

Return TRUE if the target regression coefficient matrix is good.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; psunthud@ku.edu)

Examples

```
# No example
```

`VirtualDist-class` *Distribution Objects*

Description

List of all distribution objects. These distribution objects can be used to specify random parameters or specify a marginal distribution of a variable.

Details

These distribution objects can be used to specify random parameters or marginal distribution of variables in Gaussian copula. The random parameter feature is to make data generation parameters different across replications in a simualtion study. The distribution object can be specified as random parameters in `bind`, `binds`, and `sim` (in `n`, `pmMCAR`, and `pmMAR`). The distribution object can also be used for specifying marginal distribution of factors, measurement errors, or indicators. See the data distribution object, `simDataDist`, for how to model marginal distribution of variables, which will be used in nonnormal data generation.

Distributions

Here is the list of all distribution objects and the link to their constructors.

- `simBeta` Beta Distribution
- `simBinom` Binomial Distribution
- `simCauchy` Cauchy Distribution
- `simChisq` Chi-squared Distribution
- `simExp` Exponential Distribution
- `simF` F Distribution
- `simGamma` Gamma Distribution
- `simGeom` Geometric Distribution
- `simHyper` Hypergeometric Distribution
- `simLnorm` Log Normal Distribution
- `simLogis` Logistic Distribution
- `simNbinom` Negative Binomial Distribution
- `simNorm` Normal Distribution
- `simPois` Poisson Distribution
- `simT` t Distribution
- `simUnif` Uniform Distribution
- `simWeibull` Weibull Distribution

Methods

`run` Create a random number from the specified distribution.

`summary` Summarize information withinin the object.

`summaryShort` Summarize information within the object in a short form.

`plotDist` Plot a distribution of the distribution object. Arguments: `xlim` is the range of plotting values (should be a vector of two values: lower and upper bound), `reverse` is to mirror the distribution, such as changing chi-square distribution from positively skew to negatively skew.

`skew` Find a skewness of a distribution.

`kurtosis` Find an excessive kurtosis of a distribution.

`toFunction` Export the distribution object to a function command in text that can be evaluated directly.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

See Also

List of all distribution objects.

- `SimBeta` Beta Distribution
- `SimBinom` Binomial Distribution
- `SimCauchy` Cauchy Distribution
- `SimChisq` Chi-squared Distribution
- `SimExp` Exponential Distribution
- `SimF` F Distribution
- `SimGamma` Gamma Distribution
- `SimGeom` Geometric Distribution
- `SimHyper` Hypergeometric Distribution
- `SimLnorm` Log Normal Distribution
- `SimLogis` Logistic Distribution
- `SimNbinom` Negative Binomial Distribution
- `SimNorm` Normal Distribution
- `SimPois` Poisson Distribution
- `SimT` t Distribution
- `SimUnif` Uniform Distribution
- `SimWeibull` Weibull Distribution

Here are the list of possible applications of a distribution object

- `bind` Random parameter matrix. A distribution object can be used to create random parameter.
- `binds` Random parameter symmetric matrix. A distribution object can be used to create random parameter.
- `sim` Result object that saves the result of a simulation study. A distribution object can be used to vary sample size (`n`), proportion completely missing at random (`pmMCAR`), or proportion missing at random (`pmMAR`), which make those factors (e.g., sample size) different across replications.
- `simDataDist` Data distribution object. A distribution object can be used to specify marginal distributions of variables (which can be factors, measurement errors, or indicators).

Examples

```
showClass("VirtualDist")
u1 <- simUnif(0, 1)
chi3 <- simChisq(3)
summary(chi3)
skew(chi3)
kurtosis(chi3)
plotDist(chi3)
plotDist(chi3, reverse=TRUE)
```

`weightedMean` *Calculate the weighted mean of a variable*

Description

Calculate the weighted mean of a variable

Usage

```
weightedMean(x, weight=NULL)
```

Arguments

`x`	A target vector to be averaged
`weight`	The weight of each element

Value

A weighted mean value

Author(s)

Sunthud Pornprasertmanit (University of Kansas; psunthud@ku.edu)

Examples

```
# This function is not public

# weightedMean(1:5, c(1,1,1,1,2))
```

whichMonotonic *Extract a part of a vector that is monotonically increasing or decreasing*

Description

Extract a part of a vector that is monotonically increasing or decreasing. This function will go to the anchor value and extract the neighbor values that are monotonically increasing or decreasing.

Usage

```
whichMonotonic(vec, ord=NULL, anchor=NULL)
```

Arguments

vec	The target vector to be extracted
ord	The names of each element of the vector to be attached
anchor	The position of the element to be anchored. The default value is the middle position.

Value

The monotonic part of a vector

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>)

Examples

```
# This is a private function.

# whichMonotonic(c(3, 4, 1, 2, 3, 5, 2, 1))
```

Index