Package 'simsem'

June 22, 2012

Type Package

Title SIMulated Structural Equation Modeling.

Version 0.2-7

Date 2012-06-18

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

Maintainer Sunthud Pornprasertmanit <psunthud@ku.edu>, Patrick Miller <patr1ckm@ku.edu>, Alexander Schoemann <schoemann@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 https://github.com/simsem/simsem/wiki

R topics documented:

adjust 5
anova 7
blankParameters 8
centralMoment 9
checkInputValue 9
clean 10
cleanSimResult 11
collapseExo 11
combineLatentCorExoEndo 12
combineLoadingExoEndo 13
combineMeasurementErrorExoEndo 13
combineObject 14
combinePathExoEndo 15

constantVector . . . 16
constrainMatrices . . . 16
continuousPower . . . 17
countFreeParameters . . . 19
countMACS . . . 20
cov2corMod . . . 20
createData . . . 21
createFreeParameters . . . 21
createImpliedMACS . . . 22
defaultStartingValues . . . 23
divideObject . . . 24
drawParameters . . . 25
drawParametersMisspec . . . 25
expandMatrices . . . 26
extract . . . 27
extractLavaanFit . . . 28
extractMatrixNames . . . 28
extractOpenMxFit . . . 29
extractVectorNames . . . 29
fillParam . . . 30
find2Dhist . . . 31
findFactorIntercept . . . 31
findFactorMean . . . 32
findFactorResidualVar . . . 33
findFactorTotalCov . . . 34
findFactorTotalVar . . . 36
findIndIntercept . . . 37
findIndMean . . . 38
findIndResidualVar . . . 39
findIndTotalVar . . . 40
findphist . . . 41
findPossibleFactorCor . . . 41
findPower . . . 42
findRecursiveSet . . . 43
findRowZero . . . 44
findTargetPower . . . 45
fitMeasuresChi . . . 46
freeVector . . . 46
getCondQtile . . . 47
getCutoff . . . 48
getCutoffNested . . . 49
getCutoffNonNested . . . 51
getKeywords . . . 53
getPopulation . . . 54
getPower . . . 55
getPowerFit . . . 57
getPowerFitNested . . . 59
getPowerFitNonNested . . . 61
imposeMissing . . . 63
indProd . . . 65
interpolate . . . 66
isCorMatrix . . . 67

isDefault . . . 68
isMeanConstraint . . . 68
isNullObject . . . 69
isRandom . . . 70
isVarianceConstraint . . . 71
kStat . . . 71
kurtosis . . . 72
likRatioFit . . . 73
loadingFromAlpha . . . 74
makeLabels . . . 75
matchKeywords . . . 76
MatrixSet-class . . . 77
miPool . . . 78
miPoolChi . . . 79
miPoolVector . . . 80
multipleAllEqual . . . 82
Nullclass . . . 82
overlapHist . . . 83
ParameterSet . . . 84
plot3DQtile . . . 85
plotCutoff . . . 86
plotCutoffNested . . . 87
plotCutoffNonNested . . . 89
plotDist . . . 90
plotIndividualScatter . . . 91
plotLogisticFit . . . 92
plotMisfit . . . 93
plotOverHist . . . 94
plotPower . . . 95
plotPowerFit . . . 96
plotPowerFitDf . . . 98
plotPowerFitNested . . . 99
plotPowerFitNonNested . . . 101
plotPowerSig . . . 103
plotQtile . . . 104
plotScatter . . . 105
popDiscrepancy . . . 106
popMisfit . . . 107
popMisfitMACS . . . 108
predProb . . . 109
printIfNotNull . . . 110
pValue . . . 111
pValueCondCutoff . . . 113
pValueNested . . . 114
pValueNonNested . . . 115
pValueVariedCutoff . . . 117
reassignNames . . . 118
reduceConstraint . . . 119
reduceMatrices . . . 120
residualCovariate . . . 120
revText . . . 121
run . . . 122

runFit . . . 123
runFitParam . . . 126
runLavaan . . . 127
runMI . . . 128
runMisspec . . . 129
runRep . . . 131
setOpenMxObject . . . 132
setPopulation . . . 133
simBeta . . . 134
simBinom . . . 134
simCauchy . . . 135
simChisq . . . 136
simData . . . 137
SimData-class . . . 139
simDataDist . . . 140
SimDataDist-class . . . 141
SimDataOut-class . . . 143
simEqualCon . . . 144
SimEqualCon-class . . . 146
simExp . . . 147
simF . . . 148
simFunction . . . 148
SimFunction-class . . . 150
simGamma . . . 152
SimGenLabels-class . . . 153
simGeom . . . 154
simHyper . . . 155
simLnorm . . . 155
simLogis . . . 156
simMatrix . . . 157
SimMatrix-class . . . 158
simMissing . . . 159
SimMissing-class . . . 161
SimMisspec-class . . . 162
simMisspecCFA . . . 164
simMisspecPath . . . 166
simMisspecSEM . . . 167
simModel . . . 169
SimModel-class . . . 170
SimModelMIOut-class . . . 172
SimModelOut-class . . . 173
simNbinom . . . 175
simNorm . . . 175
SimParam-class . . . 176
simParamCFA . . . 177
simParamPath . . . 179
simParamSEM . . . 180
simPois . . . 182
SimREqualCon-class . . . 183
simResult . . . 184
SimResult-class . . . 186
simResultParam . . . 188

SimResultParam-class . . . 189
SimSet-class . . . 190
simSetCFA . . . 192
simSetPath . . . 194
simSetSEM . . . 196
simT . . . 200
simUnif . . . 200
simVector . . . 201
SimVector-class . . . 202
simWeibull . . . 203
skew . . . 204
sortList . . . 204
standardize . . . 205
startingValues . . . 206
subtractObject . . . 207
summaryFit . . . 208
summaryMisspec . . . 210
summaryParam . . . 211
summaryPopulation . . . 213
summaryShort . . . 214
symMatrix . . . 215
SymMatrix-class . . . 216
tagHeaders . . . 217
toFunction . . . 218
toSimSet . . . 218
twoTailedPValue . . . 219
validateCovariance . . . 220
validateObject . . . 220
validatePath . . . 221
vectorizeObject . . . 221
VirtualDist-class . . . 222
weightedMean . . . 225
whichMonotonic . . . 225
writeLavaanCode . . . 226
writeLavaanConstraint . . . 227
writeLavaanIndividualConstraint . . . 227
writeLavaanNullCode . . . 228

Index **229**

`adjust` *Change an element in* `SimMatrix`, `SymMatrix`, *or* `SimVector`.

Description

This function will adjust an element in `SimMatrix`, `SymMatrix`, or `SimVector`. The specified element may be set to be free parameter with number or distribution object as starting values. Alternatively, the element can be fixed to be a value (such as 0).

Usage

```
adjust(target, value, pos, numAsFixed)
```

Arguments

target	Target SimMatrix, SymMatrix, or SimVector that you would like to adjust.
value	The name of distribution object that you would like to specify (put as **character** with single or double quotation) or number that represents fixed values or starting values.
pos	The position of element that you would like to adjust, such as "c(1,2)" for the element in Row 1 and Column 2 in the specified matrix.
numAsFixed	This argument is used when the VirtualDist argument was specified as number. If TRUE (as default), the number is treated as fixed parameters. If FALSE, the number is treated as a starting value and the element is set to be free parameter.

Value

Return the input SimMatrix, SymMatrix, or SimVector with adjusted element.

Note

For SymMatrix class, above- and below-diagonal elements will be adjusted simultaneously. Either above- or below-diagonal element is specified in the pos argument.

Author(s)

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

See Also

- SimMatrix for random parameter matrix
- SymMatrix for symmetric random parameter matrix
- SimVector for random parameter vector

Examples

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

u34 <- simUnif(0.3, 0.4)
LX <- adjust(LX, "u34", c(2, 1))
summary(LX)
run(LX)

LX <- adjust(LX, 0, c(2,1))
LX <- adjust(LX, 0.5, c(2,2), FALSE)
summary(LX)
run(LX)

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

```
summary(AL)

n01 <- simNorm(0, 1)
AL <- adjust(AL, "n01", 2)
run(AL)
summary(AL)
```

---

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 `SimModelOut`, each column means

- `summary`: The fit indices of each output
- `diff`: The differences in fit indices among outputs

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 <- simMatrix(loading1, 0.7)
LX2 <- simMatrix(loading2, 0.7)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model1 <- simSetCFA(LY = LX1, RPS = RPH, RTE = RTD)
CFA.Model2 <- simSetCFA(LY = LX2, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model1, 500)
SimModel1 <- simModel(CFA.Model1)
SimModel2 <- simModel(CFA.Model2)
# 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 <- simResult(5, SimData, SimModel1, seed=123567)
Output2 <- simResult(5, SimData, SimModel2, seed=123567)
anova(Output1, Output2)

Output1b <- simResult(NULL, SimData, SimModel1, seed=123567, n=seq(50, 500, 50))
Output2b <- simResult(NULL, SimData, SimModel2, seed=123567, n=seq(50, 500, 50))
anova(Output1b, Output2b)
```

---

blankParameters *Change all elements in the non-null objects to be all NAs.*

Description

Change all elements in the non-null objects to be all NAs.

Usage

```
blankParameters(object)
```

Arguments

object	The target VirtualRSet class

Value

The target object that all elements are NA

Author(s)

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

Examples

```
# No example
```

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)
```

checkInputValue *Check the value argument in the matrix, symmetric matrix, or vector objects*

Description

Check the value argument in the matrix, symmetric matrix, or vector objects constructors (simMatrix, symMatrix, and simVector) and provide back the appropriate text, which can be evaluated later directly. The checkInputValue function is used for a single value. The checkInputValueVector function is used for multiple values.

Usage

```
checkInputValue(txt)
checkInputValueVector(txt)
```

Arguments

txt	The value that is fed in the constructors

Value

The appropriate text of code that can be evaluated later

Author(s)

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

See Also

The other functions that used this function: `simMatrix`, `symMatrix`, `simVector`, and `adjust`

Examples

```
# No example
```

`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
```

collapseExo *Collapse all exogenous variables and put all in endogenous side only.*

Description

Collapse all exogenous variables and put all in endogenous side only.

Usage

```
collapseExo(object, value = 0, label = FALSE)
```

Arguments

object	The target VirtualRSet that has exogenous side.
value	The value to be filled in non-specified elements in combining matrices together
label	Keep row and column names if TRUE.

Value

The combined VirtualRSet class

Author(s)

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

Examples

```
# No example
```

`combineLatentCorExoEndo` *Combine exogenous factor correlation and endogenous factor correlation into a single matrix*

Description

Combine exogenous factor correlation, `PH`, and endogenous factor correlation, `PS`, into a single matrix

Usage

```
combineLatentCorExoEndo(PH, PS, value=0)
```

Arguments

`PH`	The exogenous factor correlation matrix
`PS`	The endogenous factor correlation matrix
`value`	The values for the correlation between exogenous and endogenous variables

Value

The combined correlation matrix

Author(s)

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

Examples

```
# No example
```

combineLoadingExoEndo *Combine factor loading from the exogenous and endogenous sides into a single matrix*

Description

Combine factor loading from the exogenous and endogenous sides into a single matrix

Usage

```
combineLoadingExoEndo(LX, LY, value = 0)
```

Arguments

LX	The factor loading matrix in the exogenous side
LY	The factor loading matrix in the endogenous side
value	the values filling in the other values in the resulting matrix

Value

The combined factor loading matrix

Author(s)

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

Examples

```
# No example
```

combineMeasurementErrorExoEndo *Combine measurement error correlation from the exogenous and endogenous sides into a single matrix*

Description

Combine measurement error correlation from the exogenous and endogenous sides into a single matrix

Usage

```
combineMeasurementErrorExoEndo(TD, TE, TH)
```

Arguments

TD	The error of measurement correlation in the exogenous side
TE	The error of measurement correlation in the endogenous side
TH	The correlation between the error of measurement in the exogenous and endogenous sides

Value

The combined error correlation matrix

Author(s)

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

Examples

```
# No example
```

`combineObject` *Combine by summing or binding two objects together.*

Description

Combine by summing or binding two objects together.

Usage

```
combineObject(object1, object2, ...)
```

Arguments

`object1`	The first object
`object2`	The second object
`...`	The additional arguments

Value

The combined object

Methods

signature(object1="SimMatrix", object2="SimMatrix") This function will combine two `SimMatrix` together by, 1) If a parameter/starting values of an element is specified, the combined object will be free parameters with the starting value. If both objects have parameter/starting values, one of `object1` will be used. 2) If both are fixed, the fixed value of `object1` will be used.

signature(object1="SimVector", object2="SimVector") This function will combine two `SimVector` together by, 1) If a parameter/starting values of an element is specified, the combined object will be free parameters with the starting value. If both objects have parameter/starting values, one of `object1` will be used. 2) If both are fixed, the fixed value of `object1` will be used.

signature(object1="vector", object2="vector") This function is used to combine two vectors. If both are null vectors, it will return null vectors. If `object2` is null vector, it will return `object1`. If both objects are not null, it will return the sum of both vectors.

signature(object1="matrix", object2="matrix") This function is used to combine two matrices. If both are null matrices, it will return null matrices. If `object2` is null matrix, it will return `object1`. If both objects are not null, it will return the sum of both objects. If both objects are correlation matrices, it will retain diagonal elements of 1. The additional argument is `correlation`, which, if `TRUE`, the diagonal elements are always fixed to 1.

signature(object1="MatrixSet", object2="MatrixSet") This function is used to combine two matrices. If both are null matrices, it will return null matrices. If object2 is null matrix, it will return `object1`. If both objects are not null, it will return the sum of both objects. If both objects are correlation matrices, it will retain diagonal elements of 1.

signature(object1="SimParam", object2="list") The `object1` is `SimParam` of that saves all free parameters and values of fixed parameters. The `object2` is the lavaan estimates or standard error. This function will find any free parameters in the `object1` and search for appropriate number from `object2` and plug in the free parameters. This function will return `SimRSet` containing parameter estimates or standard errors.

Author(s)

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

Examples

```
# No example
```

`combinePathExoEndo` *Combine the regression coefficient matrices*

Description

Combine the regression coefficient matrices both one from exogenous variables to endogenous variables and one from endogenous variables to endogenous variables

Usage

```
combinePathExoEndo(GA, BE, value = 0)
```

Arguments

`GA`	The regression coefficient matrix from exogenous variables to endogenous variables
`BE`	The regression coefficient matrix from endogenous variables to endogenous variables
`value`	The value put in the leftovers in the resulting object (such as exogenous variables –> exogenous variables)

Value

The combined regression coefficient matrix

Author(s)

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

Examples

```
# No example
```

constantVector *Create a constant vector object*

Description

Create a constant vector object SimVector

Usage

```
constantVector(constant, ni)
```

Arguments

constant	The number or character that is used to be the constant
ni	The number of items

Value

The constant vector object

Author(s)

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

Examples

```
# This function is not public.

# constantVector(0, 5)
```

constrainMatrices *Impose an equality constraint in an object*

Description

Impose an equality constraint in an object

Usage

```
constrainMatrices(object, SimEqualCon, ...)
```

Arguments

object	The desired object that would like to be constrained, such as the list of parameter matrices
SimEqualCon	The equality constraint that saves the desired constraints.
...	The additional arguments

Value

The constrained object

Methods

signature(object="MatrixSet", SimEqualCon="SimEqualCon") This function will impose constraint codes in the input object such that the number of the first element in the constraint will be copied to other elements in the constraint.

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

```
# Specify Sample Size by n
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- simMatrix(loading, 0.7)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 500)
SimModel <- simModel(CFA.Model)
# 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 <- simResult(NULL, SimData, SimModel, 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
```

---

countFreeParameters *Count how many free parameters in the target object*

Description

Count how many free parameters in the target object

Usage

```
countFreeParameters(object, ...)
```

Arguments

object	A desired object that users wish to count the number of free parameters
...	Additional arguments specific to each class

Value

The number of free parameters

Methods

signature(object="SimMatrix") Count the number of NA in free parameter matrix

signature(object="SymMatrix") Count the number of NA in upper diagonal elements in free parameter matrix

signature(object="SimVector") Count the number of NA in the elements of a free parameter vector

signature(object="SimSet") Count the number of NA in the all vector or matrix objects inside the class

signature(object="matrix") Count the number of NA in the elements in a matrix (in symmetric matrix, the lower tri part is ignored)

signature(object="vector") Count the number of NA in the elements in a vector

signature(object="VirtualRSet") Count the number of NA in the elements in all vectors and matrices inside the class

signature(object="SimEqualCon") Count the number of parameters reduced by this equality constraint

signature(object="SimREqualCon") Count the number of parameters reduced by this equality constraint

Author(s)

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

Examples

```
# No example
```

countMACS *Count the number of elements in the sufficient statistics*

Description

Count the number of elements in the sufficient statistics (mean vector and covariance matrix)

Usage

```
countMACS(object)
```

Arguments

object	The SimSet object

Value

The number of elements in the sufficient statistics

Author(s)

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

Examples

```
# No example
```

cov2corMod *Convert a covariance matrix to a correlation matrix*

Description

Convert a covariance matrix to a correlation matrix. This is the cov2cor function in the base package that takes care of the zero-variance variables

Usage

```
cov2corMod(V)
```

Arguments

V	A covariance matrix

Value

A correlation matrix

Author(s)

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

Examples

```
# No example
```

`createData` *Create data from model parameters*

Description

Create data from model parameters. The data generation can be multivariate normal, copula model, or Bollen-Stine bootstrap model.

Usage

```
createData(paramSet, n, object, dataOnly)
```

Arguments

`paramSet`	The list of parameters used for creating data. This list has one element as the actual parameters and the misspecification parameters.
`n`	Sample size
`object`	The data object, `SimData`
`dataOnly`	`TRUE` to create data in `data.frame` only. `FALSE` to create data output object, `SimDataOut`

Value

A `data.frame` or a data output object, `SimDataOut`

Author(s)

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

Examples

```
# No example
```

`createFreeParameters` *Create a free parameters object from a model specification*

Description

Create a free parameters object, `SimParam`, from a model specification, `SimSet`

Usage

```
createFreeParameters(object)
```

Arguments

`object`	The model specification in `SimSet` that saves the free parameter information

Value

The free parameter object, SimParam

Author(s)

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

Examples

```
# No comment out because this function is not public

# 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 <- simMatrix(loading, loadingValues)
# latent.cor <- matrix(NA, 2, 2)
# diag(latent.cor) <- 1
# RPH <- symMatrix(latent.cor, 0.5)
# error.cor <- matrix(0, 6, 6)
# diag(error.cor) <- 1
# RTD <- symMatrix(error.cor)
# indicator.mean <- rep(NA, 6)
# MX <- simVector(indicator.mean, 0)
# CFA.Model <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD, MX = MX)
# free <- createFreeParameters(CFA.Model)
```

---

createImpliedMACS *Create model implied mean vector and covariance matrix*

Description

Create model implied mean vector and covariance matrix from a set of parameter values

Usage

```
createImpliedMACS(object, ...)
```

Arguments

object	A matrix set containing values of parameters
...	Other objects, such as adding the parameter of model misspecification in the data output object

Value

A list with (a) M for a model-implied mean vector and (b) CM for a model-implied covariance matrix

Author(s)

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

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
LX <- simMatrix(loading, loadingValues)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, 0.5)
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTD <- symMatrix(error.cor)
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
param <- run(CFA.Model)
createImpliedMACS(param)
```

---

defaultStartingValues *Make ad hoc starting values*

Description

Make ad hoc starting values to be analyzed by the OpenMx package

Usage

```
defaultStartingValues(object)
```

Arguments

object	The SimParam object containing the analysis model

Value

The SimRSet object containing starting values

Author(s)

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

Examples

```
# No example
```

`divideObject` *Make a division on each element of the object*

Description

Make a division on each element of the object

Usage

```
divideObject(object, constant, ...)
```

Arguments

`object`	A desired object to be divided
`constant`	A divisor, which is a number
`...`	Other arguments, such as whether the matrix is a correlation matrix

Value

The divided object

Methods

signature(object="vector", constant="numeric") Divide each element in the `vector`. If the `vector` is `NullVector`, it will do nothing.

signature(object="matrix", constant="numeric") Divide each element in the `matrix`. If the `matrix` is `NullMatrix`, it will do nothing. The additional argument is `correlation`. Use `correlation=TRUE` to show that the target object is a correlation matrix and the function will keep the diagonal elements as 1.

signature(object="MatrixSet", constant="numeric") Divide all matrices and vectors in the `MatrixSet`

Author(s)

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

Examples

```
# No example
```

drawParameters *Create parameter sets (with or without model misspecification) from the data object*

Description

Create parameter sets (with or without model misspecification) from the data object, SimData

Usage

```
drawParameters(object)
```

Arguments

object	The data object, SimData, used to draw a parameter set

Value

The list of parameters with model misspecification (misspec) and without model misspecification (real)

Author(s)

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

Examples

```
# No example
```

drawParametersMisspec *Create parameter sets (with or without model misspecification) from the parameter with or without misspecification set*

Description

Create parameter sets (with or without model misspecification) from the parameter (SimSet) with or without misspecification set (SimMisspec)

Usage

```
drawParametersMisspec(objSet, objMisspec = new("NullSimMisspec"), objEqualCon = new("NullSimEqua
```

Arguments

objSet	The parameter set object, SimSet, that saves the specification of the actual parameter
objMisspec	The misspecified parameter set object, SimMisspec, that saves the specification of the misspecified parameter
objEqualCon	The equality constraint object, SimEqualCon
maxDraw	The maximum number of sample drawn. This function will draw sets of actual and misspecified parameters repeatedly until the identified sets of actual and misspecified parameters are drawn. The maximum of repetition is specified by this argument.

Value

A list of with two elements: 1) param SimRSet of real model parameters, 2) misspec SimRSet of real model parameters with model misspecification, and 3) misspecAdd the misspecification alone

Author(s)

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

Examples

```
# No example
```

---

expandMatrices *Expand the set of intercept and covariance matrices into the set of intercept/mean and covariance/correlation/variance objects*

Description

Expand the set of intercept and covariance matrices into the set of intercept/mean and covariance/correlation/variance objects

Usage

```
expandMatrices(object)
```

Arguments

object	The SimRSet class that users wish to expand

Value

The MatrixSet class containing all information from the target object

Author(s)

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

Examples

```
# No example
```

`extract` *Extract a part of an object*

Description

Extract a part of an object, such as selecting only a subset of variables from a model specification

Usage

```
extract(object, ...)
```

Arguments

`object`	The extracted object
`...`	The specification of the extracted part

Value

The extracted object

Methods

`signature(object = "vector")` The additional argument is `pos`. This is the position of the extracted vector.

`signature(object = "matrix")` The additional arguments are `row` and `col`. These are the positions of row and column. This method will prevent the matrix transforming to a vector if the number of rows or columnes is 1.

`signature(object = "data.frame")` Extract elements from a data.frame. There are several additional arguments. First, if `yOnly` is `TRUE`, then the result will provide only Y side. Second, `y` is the index of indicators in Y side to be extracted. Third, `e` is the index of factors in Y side to be extracted. Fourth, `x` is the index of the indicators in X side to be extracted. Fifth, `k` is the index of the factors in X side to be extracted. Finally, `keepOriginalName` is to not reorder the extracted variables.

Author(s)

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

See Also

This is the list of classes that can use the `extract` method.

- `SimMatrix`
- `SimVector`
- `SimSet`
- `SimDataDist`

Examples

```
extract(1:10, c(4, 5))
extract(diag(3), 1, 2:3)
```

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
```

extractMatrixNames *Extract a vector of parameter names based on specified rows and columns*

Description

Extract a vector of parameter names based on specified rows and columns

Usage

```
extractMatrixNames(columnName, keepRow=NULL, keepCol=NULL)
```

Arguments

columnName	A column name representing matrix elements (e.g., LY2_1) that will be used to be extracted
keepRow	The rows of the matrix that we need to keep
keepCol	The columns of the matrix that we need to keep

Value

A list that contains: 1) columnName: Original column name and 2) newName: Reordered column name.

Author(s)

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

Examples

```
# The function is not public

# vec <- c("LY1_1", "LY2_1", "LY3_1", "LY4_2", "LY5_2", "LY6_2", "LY7_3")
# extractMatrixNames(vec, 5:6, 2)
```

---

extractOpenMxFit *Extract the fit indices reported by the* OpenMx *result*

Description

Extract the fit indices reported by the OpenMx result

Usage

```
extractOpenMxFit(indiv.result)
```

Arguments

indiv.result	The OpenMx object

Value

A vector of fit indices

Author(s)

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

Examples

```
# No example
```

---

extractVectorNames *Extract a vector of parameter names based on specified elements*

Description

Extract a vector of parameter names based on specified elements

Usage

```
extractVectorNames(columnName, keep=NULL)
```

Arguments

columnName	A name representing vector elements (e.g., TY2) that will be used to be extracted
keep	The elements of the vector that we need to keep

Value

A list that contains: 1) columnName: Original element names and 2) newName: Reordered element names.

Author(s)

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

Examples

```
# The function is not public

#vec <- c("TY1", "TY2", "TY3", "TY4", "TY5", "TY6", "TY7")
#extractVectorNames(vec, 5:6)
```

---

fillParam *Fill in other objects based on the parameter values of current objects*

Description

Fill in other objects based on the parameter values of current objects

Usage

```
fillParam(param, modelType)
```

Arguments

param	The linkS4class{MatrixSet} class that some matrices have not been filled.
modelType	Analysis model type

Value

The same object that all matrices have been filled.

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

```
# Specify Sample Size by n
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- simMatrix(loading, 0.4)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 500)
SimModel <- simModel(CFA.Model)

# Specify both sample size and percent missing completely at random
Output <- simResult(NULL, SimData, SimModel, n=seq(100, 200, 20), pmMCAR=c(0, 0.1, 0.2))
pow <- getPower(Output)
findPower(pow, "N", 0.80)
```

---

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
```

`freeVector` *Create a free parameters vector with a starting values in a vector object*

Description

Create a free parameters vector with a starting values in a vector object, SimVector

Usage

```
freeVector(start, ni)
```

Arguments

start	The starting values or free parameters of the vector object, SimVector
ni	The length of the vector object

Value

The vector object with the specified parameter values and length

Author(s)

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

Examples

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

# freeVector(0, 5)
```

---

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

```
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 <- simMatrix(loading, loadingValues)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, 0.5)
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTD <- symMatrix(error.cor)
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 200)
SimModel <- simModel(CFA.Model)
# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output <- simResult(5, SimData, SimModel)
getCutoff(Output, 0.05)

# Finding the cutoff when the sample size is varied.
Output2 <- simResult(NULL, SimData, SimModel, n=seq(50, 100, 10))
getCutoff(Output2, 0.05, nVal = 75)
```

`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

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

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

error.cor.mis <- matrix(NA, 6, 6)
diag(error.cor.mis) <- 1
RTD.Mis <- symMatrix(error.cor.mis, "n1")
CFA.Model.NULL.Mis <- simMisspecCFA(RTE = RTD.Mis)

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

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

SimData.NULL <- simData(CFA.Model.NULL, 500)

SimModel.NULL <- simModel(CFA.Model.NULL)
SimModel.ALT <- simModel(CFA.Model.ALT)

# The actual number of replications should be greater than 10.
Output.NULL.NULL <- simResult(10, SimData.NULL, SimModel.NULL)
Output.NULL.ALT <- simResult(10, SimData.NULL, SimModel.ALT)

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

`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

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

loading.A <- matrix(0, 8, 2)
loading.A[1:3, 1] <- NA
loading.A[4:8, 2] <- NA
LX.A <- simMatrix(loading.A, 0.7)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, "u79")
RTD <- symMatrix(diag(8))
CFA.Model.A <- simSetCFA(LY = LX.A, RPS = RPH, RTE = RTD)

error.cor.mis <- matrix(NA, 8, 8)
diag(error.cor.mis) <- 1
RTD.Mis <- symMatrix(error.cor.mis, "n1")
CFA.Model.A.Mis <- simMisspecCFA(RTE = RTD.Mis)

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

SimData.A <- simData(CFA.Model.A, 500)
SimData.B <- simData(CFA.Model.B, 500)

SimModel.A <- simModel(CFA.Model.A)
SimModel.B <- simModel(CFA.Model.B)

# The actual number of replications should be greater than 10.
Output.A.A <- simResult(10, SimData.A, SimModel.A)
Output.A.B <- simResult(10, SimData.A, SimModel.B)
Output.B.A <- simResult(10, SimData.B, SimModel.A)
```

```
Output.B.B <- simResult(10, SimData.B, SimModel.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)
```

`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

- `LY` Factor loading matrix between endogenous factors and Y indicators
- `TE` Covariance matrix between Y measurement error
- `RTE` Correlation matrix between Y measurement error
- `PS` Residual covariance of endogenous factors
- `RPS` Residual correlation of endogenous factors
- `BE` Regression effect among endogenous factors
- `VY` Total variance of Y indicators
- `VPS` Residual variances of endogenous factors
- `VE` Total variance of endogenous factors
- `TY` Measurement intercepts of Y indicators
- `ME` Factor means of endogenous factors
- `VTE` Variance of Y measurement error
- `AL` Factor intercepts of endogenous factors
- `MY` Total Mean of Y indicators
- `LX` Factor loading matrix between exogenous factors and X indicators
- `TD` Covariance matrix between X measurement error
- `RTD` Correlation matrix between X measurement error
- `PH` Covariance among exogenous factors
- `RPH` Correlation among exogenous factors
- `GA` Regression effect from exogenous factors to endogenous factors
- `VX` Total variance of X indicators
- `VPH` Variance of exogenous factors
- `TX` Measurement intercepts of X indicators
- `KA` Factor Mean of exogenous factors

- VTD Variance of X measurement error
- MX Total Mean of X indicators
- TH Measurement error covariance between X indicators and Y indicators
- RTH Measurement error correlation between X indicators and Y indicators
- loading Any factor loading matrix keywords
- errorCov Any measurement error covariance matrix keywords
- errorCor Any measurement error correlation matrix keywords
- errorVar Any measurement error variance vector keywords
- indicatorVar Any indicator variance vector keywords
- indicatorMean Any indicator mean vector keywords
- facCov Any factor covariance matrix keywords
- facCor Any factor correlation matrix keywords
- intercept Any intercept vector keywords
- facMean Any factor mean vector keywords
- facVar Any factor variance vector keywords
- 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{SimModelOut}, linkS4class{SimDataOut}, or 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{SimModelOut}, linkS4class{SimDataOut}, or 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

- SimDataOut for data output object
- SimModelOut for model output object
- 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

```
# Specify Sample Size by n
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- simMatrix(loading, 0.7)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 500)
SimModel <- simModel(CFA.Model)
# 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 <- simResult(NULL, SimData, SimModel, 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))
```

`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

```
loading.null <- matrix(0, 6, 1)
loading.null[1:6, 1] <- NA
LX.NULL <- simMatrix(loading.null, 0.7)
RPH.NULL <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model.NULL <- simSetCFA(LY = LX.NULL, RPS = RPH.NULL, RTE = RTD)
SimData.NULL <- simData(CFA.Model.NULL, 500)
SimModel <- simModel(CFA.Model.NULL)
# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output.NULL <- simResult(5, SimData.NULL, SimModel)
Cut.NULL <- getCutoff(Output.NULL, 0.95)

u79 <- simUnif(0.7, 0.9)
loading.alt <- matrix(0, 6, 2)
loading.alt[1:3, 1] <- NA
loading.alt[4:6, 2] <- NA
LX.ALT <- simMatrix(loading.alt, 0.7)
latent.cor.alt <- matrix(NA, 2, 2)
diag(latent.cor.alt) <- 1
RPH.ALT <- symMatrix(latent.cor.alt, "u79")
CFA.Model.ALT <- simSetCFA(LY = LX.ALT, RPS = RPH.ALT, RTE = RTD)
SimData.ALT <- simData(CFA.Model.ALT, 500)
Output.ALT <- simResult(5, SimData.ALT, SimModel)
```

```
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 <- simResult(NULL, SimData.NULL, SimModel, n=seq(50, 500, 50))
Output.ALT2 <- simResult(NULL, SimData.ALT, SimModel, n=seq(50, 500, 50))
getPowerFit(Output.ALT2, nullObject=Output.NULL2, nVal=250)
```

`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

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

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

error.cor.mis <- matrix(NA, 6, 6)
diag(error.cor.mis) <- 1
RTD.Mis <- symMatrix(error.cor.mis, "rnorm(1,0,0.1)")
CFA.Model.NULL.Mis <- simMisspecCFA(RTE = RTD.Mis)

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

# loading.alt.mis <- matrix(NA, 6, 2)
# loading.alt.mis[is.na(loading.alt)] <- 0
# LX.alt.mis <- simMatrix(loading.alt.mis, "runif(1,-.2,.2)")
# CFA.Model.alt.mis <- simMisspecCFA(LY = LX.alt.mis, RTE=RTD.Mis)

SimData.NULL <- simData(CFA.Model.NULL, 500)
SimData.ALT <- simData(CFA.Model.ALT, 500)

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

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

Output.NULL.NULL <- simResult(10, SimData.NULL, SimModel.NULL)
Output.ALT.NULL <- simResult(10, SimData.ALT, SimModel.NULL)
Output.NULL.ALT <- simResult(10, SimData.NULL, SimModel.ALT)
Output.ALT.ALT <- simResult(10, SimData.ALT, SimModel.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 <- simResult(NULL, SimData.NULL, SimModel.NULL, n=seq(50, 500, 50))
Output.ALT.NULL2 <- simResult(NULL, SimData.ALT, SimModel.NULL, n=seq(50, 500, 50))
Output.NULL.ALT2 <- simResult(NULL, SimData.NULL, SimModel.ALT, n=seq(50, 500, 50))
Output.ALT.ALT2 <- simResult(NULL, SimData.ALT, SimModel.ALT, n=seq(50, 500, 50))

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), nVal = 250)
```

`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

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

loading.A <- matrix(0, 8, 2)
loading.A[1:3, 1] <- NA
loading.A[4:8, 2] <- NA
LX.A <- simMatrix(loading.A, 0.7)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, "u79")
RTD <- symMatrix(diag(8))
CFA.Model.A <- simSetCFA(LY = LX.A, RPS = RPH, RTE = RTD)

error.cor.mis <- matrix(NA, 8, 8)
diag(error.cor.mis) <- 1
RTD.Mis <- symMatrix(error.cor.mis, "n1")
CFA.Model.A.Mis <- simMisspecCFA(RTE = RTD.Mis)

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

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

SimData.A <- simData(CFA.Model.A, 500)
SimData.B <- simData(CFA.Model.B, 500)

SimModel.A <- simModel(CFA.Model.A)
SimModel.B <- simModel(CFA.Model.B)

# The actual number of replications should be greater than 10.
Output.A.A <- simResult(10, SimData.A, SimModel.A)
Output.A.B <- simResult(10, SimData.A, SimModel.B)
Output.B.A <- simResult(10, SimData.B, SimModel.A)
Output.B.B <- simResult(10, SimData.B, SimModel.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))
```

`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 = 0, twoMethod = 0,  prAttr = 0, timePoints = 1,
ignoreCols = 0, threshold = 0, logical = new("NullMatrix"))
```

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`	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.
`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 simResult 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)
```

---

indProd *Make a product of indicators using mean centering or double-mean centering*

Description

This function will use mean centering or double 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

```
indProd(data, var1, var2, match = TRUE, meanC = TRUE, residualC = FALSE, doubleMC = TRUE, namesP
```

Arguments

data	The desired data to be transformed.
var1	Names or indices of the variables loaded on the first factor
var2	Names or indices of the variables loaded on the second factor
match	Specify TRUE to use match-paired approach (Marsh, Wen, & Hau, 2004). If FALSE, the resulting products are all possible products.
meanC	Specify TRUE for mean centering the main effect indicator before making the products

residualC	Specify TRUE for residual centering the products by the main effect indicators (Little, Bovaird, & Widaman, 2006).
doubleMC	Specify TRUE for centering the resulting products (Lin et. al., 2010)
namesProd	The names of resulting products

Value

The original data attached with the products.

Author(s)

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

References

- Marsh, H. W., Wen, Z. & Hau, K. T. (2004). Structural equation models of latent interactions: Evaluation of alternative estimation strategies and indicator construction. *Psychological Methods, 9,* 275-300.
- Lin, G. C., Wen, Z., Marsh, H. W., & Lin, H. S. (2010). Structural equation models of latent interactions: Clarification of orthogonalizing and double-mean-centering strategies. *Structural Equation Modeling, 17*, 374-391.
- Little, T. D., Bovaird, J. A., & Widaman, K. F. (2006). On the merits of orthogonalizing powered and product terms: Implications for modeling interactions among latent variables. *Structural Equation Modeling, 13*, 497-519.

See Also

- simFunction to use this function within a simulation study

Examples

```
dat <- indProd(attitude[,-1], var1=1:3, var2=4:6)
```

---

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
```

---

isCorMatrix *Check whether a* matrix *is a possible correlation matrix*

Description

Check whether a matrix is a possible correlation matrix

Usage

```
isCorMatrix(matrixA)
```

Arguments

matrixA	The target matrix

Value

TRUE if the target matrix is a possible correlation matrix

Author(s)

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

Examples

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

# isCorMatrix(diag(5))
```

isDefault *Check whether a vector object is default*

Description

Check whether a specified the vector object is a default vector object such that users did not specify anything. For example, check whether means of indicators are specified as 1.

Usage

```
isDefault(object)
```

Arguments

object	The target vector object

Value

TRUE if the target vector object is a default vector object

Author(s)

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

Examples

```
# No example
```

isMeanConstraint *Check whether all rownames in a constraint matrix containing symbols of means vectors*

Description

Check whether all rownames in a constraint matrix containing symbols of means vectors

Usage

```
isMeanConstraint(Name)
```

Arguments

Name	The rownames of a constraint matrix

Value

TRUE if all rownames are means vectors

Author(s)

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

Examples

```
# No example
```

`isNullObject` *Check whether the object is the* `NULL` *type of that class*

Description

Check whether the object is the `NULL` type of that class

Usage

```
isNullObject(target)
```

Arguments

`target`	The target object

Value

`TRUE` if the model is the `NULL` type.

Methods

signature(target="vector") Check whether the `vector` is `NullVector`, `NaN`, or a vector with zero length

signature(target="matrix") Check whether the `matrix` is `NullMatrix`, `NaN`, or a matrix with 0×0 dimension

signature(target="data.frame") Check whether the `data.frame` is `NullDataFrame` or a data frame with 0 row or 0 column

signature(target="SimMatrix") Check whether the `SimMatrix` is `NullSimMatrix`

signature(target="SymMatrix") Check whether the `SymMatrix` is `NullSymMatrix`

signature(target="SimVector") Check whether the `SimVector` is `NullSimVector`

signature(target="SimSet") Check whether the `SimSet` is `NullSimSet`

signature(target="SimEqualCon") Check whether the `SimEqualCon` is `NullSimEqualCon`

signature(target="SimREqualCon") Check whether the `SimREqualCon` is `NullSimREqualCon`

signature(target="SimMisspec") Check whether the `SimMisspec` is `NullSimMisspec`

signature(target="SimMissing") Check whether the `SimMissing` is `NullSimMissing`

signature(target="SimDataDist") Check whether the `SimDataDist` is `NullSimDataDist`

signature(target="SimFunction") Check whether the `SimFunction` is `NullSimFunction`

Author(s)

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

Examples

```
# No example
```

isRandom *Check whether the object contains any random parameters*

Description

Check whether the object contains any random parameters

Usage

```
isRandom(object)
```

Arguments

object	The object to be checked

Value

TRUE if the object contains any random parameters

Methods

signature(object="SimMatrix") Check whether the SimMatrix contains any random parameters

signature(object="SimVector") Check whether the SimVector contains any random parameters

signature(object="SimSet") Check whether the SimSet contains any random parameters

signature(object="SimMisspec") Check whether the SimMisspec contains any random parameters. If the optimization method is used in the misspecification object (optMisfit="min" or optMisfit="max"), the misspecification object is said to be fixed because the optimized set will be the same across any fixed parameters.

signature(object="SimData") Check whether the SimData contains any random actual or misspecified parameters

Author(s)

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

Examples

```
# No example
```

isVarianceConstraint *Check whether all rownames in a constraint matrix containing symbols of variance vectors*

Description

Check whether all rownames in a constraint matrix containing symbols of variance vectors

Usage

```
isVarianceConstraint(Name)
```

Arguments

Name	The rownames of a constraint matrix

Value

TRUE if all rownames are variances vectors

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`	`SimModelOut` that saves the analysis result of the first model from the target dataset
`outMod2`	`SimModelOut` 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

```
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, 10, misspec=misspec)
output.A.B <- runFit(model.A, PoliticalDemocracy, 10, misspec=misspec, analyzeModel=model.B)
output.B.A <- runFit(model.B, PoliticalDemocracy, 10, misspec=misspec, analyzeModel=model.A)
output.B.B <- runFit(model.B, PoliticalDemocracy, 10, misspec=misspec)
likRatioFit(out.A, out.B, output.A.A, output.A.B, output.B.A, output.B.B)
```

---

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)
```

`makeLabels` *Make parameter names for each element in matrices or vectors or the name for the whole object*

Description

Make parameter names for each element in matrices or vectors or the name for the whole object

Usage

```
makeLabels(object, ...)
```

Arguments

`object`	The target object to find labels
`...`	Additional arguments

Value

The labeled object.

Methods

signature(object="vector") There are two additional arguments in this function: 1) `name` and `package`. The `name` is the name of this object. The `package` is the style to build the labels. This function will create labels of each element by the object name followed by the number of elements in a vector.

signature(object="matrix") There are two additional arguments in this function: 1) `name` and `package`. The `name` is the name of this object. The `package` is the style to build the labels. This function will create labels of each element by the object name followed by the numbers of rows and columns in a matrix.

signature(object="SimParam") The only addition argument is `package`, which is the style to build the labels. This function will create labels of each element by the object name followed by the numbers of rows and columns (or the number of elements) in a matrix in every matrix (or a vector) in the free parameter object.

signature(object="VirtualDist") This function will create a description of attributes. The additional argument is `digit`, which is the number of decimals.

signature(object="SimSet") The only addition argument is `package`, which is the style to build the labels. This function will create labels of each element by the object name followed by the numbers of rows and columns (or the number of elements) in a matrix in every matrix (or a vector) in the free parameter object.

Author(s)

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

Examples

```
# No example
```

`matchKeywords` *Search for the keywords and check whether the specified text match one in the name vector*

Description

Search for the keywords and check whether the specified text match one in the name vector

Usage

```
matchKeywords(Names, keywords)
```

Arguments

`Names`	The name of the searching object
`keywords`	The name of the keywords vector that would like to matched

Value

The position of keywords in the vector. Return `0` if the names does not match the specified vector.

Author(s)

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

Examples

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

# matchKeywords("ly", c("LY", "LX"))
```

MatrixSet-class *Class* "MatrixSet"

Description

Set of vectors and matrices that saves model specification (CFA, Path analysis, or SEM) for data generation or trivial model misspecification for data generation.

Slots

`modelType`: Model type (CFA, Path, or SEM)

`LY`: Factor loading matrix between endogenous factors and Y indicators

`TE`: Covariance matrix between Y measurement error

`RTE`: Correlation matrix between Y measurement error

`VTE`: Variance of Y measurement error

`PS`: Residual covariance of endogenous factors

`RPS`: Residual correlation of endogenous factors

`VPS`: Residual variances of endogenous factors

`BE`: Regression effect among endogenous factors

`TY`: Measurement intercepts of Y indicators

`AL`: Factor intercepts of endogenous factors

`ME`: Factor means of endogenous factors

`MY`: Total Mean of Y indicators

`VE`: Total variance of endogenous factors

`VY`: Total variance of Y indicators

`LX`: Factor loading matrix between exogenous factors and X indicators

`TD`: Covariance matrix between X measurement error

`RTD`: Correlation matrix between X measurement error

`VTD`: Variance of X measurement error

`PH`: Covariance among exogenous factors

`RPH`: Correlation among exogenous factors

`GA`: Regreeion effect from exogenous factors to endogenous factors

`TX`: Measurement intercepts of X indicators

`KA`: Factor Mean of exogenous factors

`MX`: Total Mean of X indicators

`VPH`: Variance of exogenous factors

`VX`: Total variance of X indicators

`TH`: Measurement error covariance between X indicators and Y indicators

`RTH`: Measurement error correlation between X indicators and Y indicators

Methods

summary Get the summary of model specification

Author(s)

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

See Also

- SimSet
- SimMisspec.

Examples

```
showClass("SimSet")

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 <- simMatrix(loading, loadingValues)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, 0.5)
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTD <- symMatrix(error.cor)
CFA.Model <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD)
MatrixSet <- run(CFA.Model)
summary(MatrixSet)
```

---

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 for combining results in the SimModelOut format.

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)
```

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)
```

Nullclass *Null Objects*

Description

List of all null objects.

Distributions

Here is the list of all null objects and the link to their original classes.

- `NullDataFrame` The null class from the `data.frame` class
- `NullVector` The null class from the `vector` class
- `NullMatrix` The null class from the `matrix` class
- `NullSimMatrix` The null class from the `SimMatrix` class
- `NullSymMatrix` The null class from the `SymMatrix` class
- `NullSimVector` The null class from the `SimVector` class
- `NullSimSet` The null class from the `SimSet` class
- `NullSimEqualCon` The null class from the `SimEqualCon` class
- `NullSimREqualCon` The null class from the `SimREqualCon` class

- NullRSet The null class from the VirtualRSet class
- NullSimMisspec The null class from the SimMisspec class
- NullSimDataDist The null class from the SimDataDist class
- NullSimMissing The null class from the SimMissing class
- NullSimFunction The null class from the SimFunction class

Author(s)

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

Examples

```
# No example
```

---

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")
```

ParameterSet *Class* "VirtualRSet", "SimLabels" *and* "SimRSet"

Description

"SimRSet" is the set of vectors and matrices arrangements that will save values that will be used for various purposes. "SimLabels" is set of vectors and matrices arrangements that will save labels that will be used for various purposes. "VirtualRSet" is the superclass of the "SimRSet", SimLabels, and SimParam.

Slots

modelType: Model type (CFA, Path, or SEM)

LY: Factor loading matrix between endogenous factors and Y indicators

TE: Covariance matrix between Y measurement error

PS: Residual covariance of endogenous factors

BE: Regression effect among endogenous factors

TY: Measurement intercepts of Y indicators

AL: Factor intercepts of endogenous factors

LX: Factor loading matrix between exogenous factors and X indicators

TD: Covariance matrix between X measurement error

PH: Covariance among exogenous factors

GA: Regreeion effect from exogenous factors to endogenous factors

TX: Measurement intercepts of X indicators

KA: Factor Mean of exogenous factors

TH: Measurement error covariance between X indicators and Y indicators

Methods

summary Get the summary of model specification.

Author(s)

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

See Also

- SimParam

Examples

```
# No 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

```
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 <- simMatrix(loading, loadingValues)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, 0.5)
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTD <- symMatrix(error.cor)
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 200)
SimModel <- simModel(CFA.Model)
# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output <- simResult(5, SimData, SimModel)
plotCutoff(Output, 0.05, usedFit=c("RMSEA", "SRMR", "CFI", "TLI"))

# Varying N
Output2 <- simResult(NULL, SimData, SimModel, n=seq(450, 500, 10))
plotCutoff(Output2, 0.05)

# Varying N and pmMCAR
Output3 <- simResult(NULL, SimData, SimModel, n=seq(450, 500, 10), pmMCAR=c(0, 0.05, 0.1, 0.15))
plotCutoff(Output3, 0.05)
```

`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

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

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

error.cor.mis <- matrix(NA, 6, 6)
diag(error.cor.mis) <- 1
RTD.Mis <- symMatrix(error.cor.mis, "n1")
CFA.Model.NULL.Mis <- simMisspecCFA(RTE = RTD.Mis)

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

SimData.NULL <- simData(CFA.Model.NULL, 500)

SimModel.NULL <- simModel(CFA.Model.NULL)
SimModel.ALT <- simModel(CFA.Model.ALT)

# The actual number of replications should be greater than 10.
Output.NULL.NULL <- simResult(10, SimData.NULL, SimModel.NULL)
Output.NULL.ALT <- simResult(10, SimData.NULL, SimModel.ALT)

plotCutoffNested(Output.NULL.NULL, Output.NULL.ALT, alpha=0.05)
```

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

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

loading.A <- matrix(0, 8, 2)
loading.A[1:3, 1] <- NA
loading.A[4:8, 2] <- NA
LX.A <- simMatrix(loading.A, 0.7)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, "u79")
RTD <- symMatrix(diag(8))
CFA.Model.A <- simSetCFA(LY = LX.A, RPS = RPH, RTE = RTD)

error.cor.mis <- matrix(NA, 8, 8)
diag(error.cor.mis) <- 1
RTD.Mis <- symMatrix(error.cor.mis, "n1")
CFA.Model.A.Mis <- simMisspecCFA(RTE = RTD.Mis)

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

SimData.A <- simData(CFA.Model.A, 500)
SimData.B <- simData(CFA.Model.B, 500)

SimModel.A <- simModel(CFA.Model.A)
SimModel.B <- simModel(CFA.Model.B)

# The actual number of replications should be greater than 10.
Output.A.A <- simResult(10, SimData.A, SimModel.A)
Output.A.B <- simResult(10, SimData.A, SimModel.B)
Output.B.A <- simResult(10, SimData.B, SimModel.A)
Output.B.B <- simResult(10, SimData.B, SimModel.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)
```

---

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 parameter result object, SimResultParam
`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

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

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

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

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

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

mis.path.BE <- matrix(0, 4, 4)
mis.path.BE[4, 1:2] <- NA
mis.BE <- simMatrix(mis.path.BE, "u1")
Path.Mis.Model <- simMisspecPath(BE = mis.BE, misfitType="rmsea") #, misfitBound=c(0.05, 0.08))

# The number of replications in actual analysis should be much more than 5
ParamObject <- simResultParam(20, Path.Model, Path.Mis.Model)
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

```
# Specify Sample Size by n
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- simMatrix(loading, 0.4)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 500)
SimModel <- simModel(CFA.Model)
# 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 <- simResult(NULL, SimData, SimModel, n=seq(100, 200, 20), pmMCAR=c(0, 0.1, 0.2))
plotPower(Output, "LY1_1", contMCAR=FALSE)
```

---

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

```
loading.null <- matrix(0, 6, 1)
loading.null[1:6, 1] <- NA
LX.NULL <- simMatrix(loading.null, 0.7)
RPH.NULL <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model.NULL <- simSetCFA(LY = LX.NULL, RPS = RPH.NULL, RTE = RTD)
SimData.NULL <- simData(CFA.Model.NULL, 500)
SimModel <- simModel(CFA.Model.NULL)
# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
```

```
Output.NULL <- simResult(5, SimData.NULL, SimModel)

loading.alt <- matrix(0, 6, 2)
loading.alt[1:3, 1] <- NA
loading.alt[4:6, 2] <- NA
LX.ALT <- simMatrix(loading.alt, 0.7)
latent.cor.alt <- matrix(NA, 2, 2)
diag(latent.cor.alt) <- 1
RPH.ALT <- symMatrix(latent.cor.alt, 0.5)
CFA.Model.ALT <- simSetCFA(LY = LX.ALT, RPS = RPH.ALT, RTE = RTD)
SimData.ALT <- simData(CFA.Model.ALT, 500)
Output.ALT <- simResult(5, SimData.ALT, SimModel)
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"))
```

`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

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

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

error.cor.mis <- matrix(NA, 6, 6)
diag(error.cor.mis) <- 1
RTD.Mis <- symMatrix(error.cor.mis, "rnorm(1,0,0.1)")
```

```
CFA.Model.NULL.Mis <- simMisspecCFA(RTE = RTD.Mis)

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

# loading.alt.mis <- matrix(NA, 6, 2)
# loading.alt.mis[is.na(loading.alt)] <- 0
# LX.alt.mis <- simMatrix(loading.alt.mis, "runif(1,-.2,.2)")
# CFA.Model.alt.mis <- simMisspecCFA(LY = LX.alt.mis, RTE=RTD.Mis)

SimData.NULL <- simData(CFA.Model.NULL, 500)
SimData.ALT <- simData(CFA.Model.ALT, 500)

SimModel.NULL <- simModel(CFA.Model.NULL)
SimModel.ALT <- simModel(CFA.Model.ALT)

Output.NULL.NULL <- simResult(10, SimData.NULL, SimModel.NULL)
Output.ALT.NULL <- simResult(10, SimData.ALT, SimModel.NULL)
Output.NULL.ALT <- simResult(10, SimData.NULL, SimModel.ALT)
Output.ALT.ALT <- simResult(10, SimData.ALT, SimModel.ALT)

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

Output.NULL.NULL2 <- simResult(NULL, SimData.NULL, SimModel.NULL, n=seq(50, 500, 50))
Output.ALT.NULL2 <- simResult(NULL, SimData.ALT, SimModel.NULL, n=seq(50, 500, 50))
Output.NULL.ALT2 <- simResult(NULL, SimData.NULL, SimModel.ALT, n=seq(50, 500, 50))
Output.ALT.ALT2 <- simResult(NULL, SimData.ALT, SimModel.ALT, n=seq(50, 500, 50))

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)
```

`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

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

loading.A <- matrix(0, 8, 2)
loading.A[1:3, 1] <- NA
loading.A[4:8, 2] <- NA
LX.A <- simMatrix(loading.A, 0.7)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, "u79")
RTD <- symMatrix(diag(8))
CFA.Model.A <- simSetCFA(LY = LX.A, RPS = RPH, RTE = RTD)

error.cor.mis <- matrix(NA, 8, 8)
diag(error.cor.mis) <- 1
RTD.Mis <- symMatrix(error.cor.mis, "n1")
CFA.Model.A.Mis <- simMisspecCFA(RTE = RTD.Mis)

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

SimData.A <- simData(CFA.Model.A, 500)
SimData.B <- simData(CFA.Model.B, 500)

SimModel.A <- simModel(CFA.Model.A)
SimModel.B <- simModel(CFA.Model.B)

# The actual number of replications should be greater than 10.
Output.A.A <- simResult(10, SimData.A, SimModel.A)
Output.A.B <- simResult(10, SimData.A, SimModel.B)
Output.B.A <- simResult(10, SimData.B, SimModel.A)
Output.B.B <- simResult(10, SimData.B, SimModel.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))
```

---

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)
```

---

popMisfit *Calculate population misfit*

Description

Calculate population misfit given the types of object

Usage

```
popMisfit(param, misspec, dfParam=NULL, fit.measures="all", ...)
```

Arguments

`param`	The object represents the actual parameters
`misspec`	The object represents the misspecified parameters
`dfParam`	The degree of freedoms of the target model
`fit.measures`	The type of population misfit measures: `"F0"`, `"rmsea"`, or `"srmr"`. See `popMisfitMACS` for further details.
`...`	The additional arguments

Value

The value of population misfit

Methods

signature(param = "matrix", misspec = "matrix") Calculate population misfit using actual covariance matrices with and without misspecification

signature(param = "list", misspec = "list") Calculate population misfit from two lists of sufficient statistics with and without misspecification. The list should have the first argument as the mean vector and the second argument as the covariance matrix.

signature(param = "SimRSet", misspec = "SimRSet") Calculate population misfit from two set of parameters with and without misspecification.

signature(param = "MatrixSet", misspec = "MatrixSet") Calculate population misfit from two set of parameters with and without misspecification.

signature(param = "SimSet", misspec = "SimMisspec") Calculate population misfit from the set of actual parameter specification and the set of misspecified parameter specification.

Author(s)

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

Examples

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

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

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

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

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

mis.path.BE <- matrix(0, 4, 4)
mis.path.BE[4, 1:2] <- NA
mis.BE <- simMatrix(mis.path.BE, "u1")
Path.Mis.Model <- simMisspecPath(BE = mis.BE, misfitType="rmsea") #, misfitBound=c(0.05, 0.08))

popMisfit(Path.Model, Path.Mis.Model, fit.measures="rmsea")
```

---

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 SimModelOut) 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).

signature(target="SimModelOut", dist="SimResult") This method will find the *p* value of the analysis result compared to the simulated sampling distribution in a result object (SimResult). The additional arguments are usedFit, nVal, pmMCARval, pmMARval, and df. The usedFit is the vector of names of fit indices that researchers wish to find the *p* value from. 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

- SimModelOut to see how to get the analysis result of observed data
- SimResult to run a simulation study
- runFit to run a simulation study based on the parameter estimates from the analysis result of observed data

Examples

```
# 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)
```

`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	SimModelOut that saves the analysis result of the nested model from the target dataset
outParent	SimModelOut 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

- SimModelOut to see how to get the analysis result of observed data
- SimResult to run a simulation study
- runFit to run a simulation study based on the parameter estimates from the analysis result of observed data

Examples

```
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)
```

---

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	SimModelOut that saves the analysis result of the first model from the target dataset
outMod2	SimModelOut 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

- SimModelOut to see how to get the analysis result of observed data
- SimResult to run a simulation study
- runFit to run a simulation study based on the parameter estimates from the analysis result of observed data

Examples

```
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, 10, misspec=misspec)
output.A.B <- runFit(model.A, PoliticalDemocracy, 10, misspec=misspec, analyzeModel=model.B)
output.B.A <- runFit(model.B, PoliticalDemocracy, 10, misspec=misspec, analyzeModel=model.A)
output.B.B <- runFit(model.B, PoliticalDemocracy, 10, misspec=misspec)
pValueNonNested(out.A, out.B, output.A.A, output.A.B, output.B.A, output.B.B)
```

---

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
```

---

reassignNames *Reassign the name of equality constraint*

Description

Match the rownames of the equality constraint, check whether it match any model matrices, and substitute with an appropriate matrix name.

Usage

```
reassignNames(modelType, Name)
```

Arguments

modelType	The type of the analysis model
Name	The row names of a matrix in the equality constraint

Value

The appropriate row names given the analysis model

Author(s)

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

Examples

```
# No example
```

`reduceConstraint` *Reduce the model constraint to data generation parameterization to analysis model parameterization.*

Description

Reduce the model constraint to data generation parameterization (in `SimSet`) to analysis model parameterization (in `linkS4class{SimRSet}`). Some symbols will be reduced to appropriate position such as `VTE[3]` to `TE[3, 3]`

Usage

```
reduceConstraint(SimEqualCon)
```

Arguments

`SimEqualCon`	The equality constraint object, `SimEqualCon`

Value

The reduced equality constraint object, `SimREqualCon`

Author(s)

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

Examples

```
# No example
```

reduceMatrices — *Reduce the model constraint to data generation parameterization to analysis model parameterization.*

Description

Reduce the MatrixSet class with the data generation parameterization to the analysis model parameterization and save it in the SimRSet object

Usage

```
reduceMatrices(object)
```

Arguments

object	The target MatrixSet object

Value

The reduced parameter values in the SimRSet object

Author(s)

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

Examples

```
# No example
```

residualCovariate — *Residual centered all target indicators by covariates*

Description

This function will regress target variables on the covariate and replace the target variables by the residual of the regression analysis. This procedure is useful to control the covariate from the analysis model (Geldhof, Pornprasertmanit, Schoemann, & Little, in press).

Usage

```
residualCovariate(data, targetVar, covVar)
```

Arguments

data	The desired data to be transformed.
targetVar	Varible names or the position of indicators that users wish to be residual centered (as dependent variables)
covVar	Covariate names or the position of the covariates using for residual centering (as independent variables) onto target variables

Value

The data that the target variables replaced by the residuals

Author(s)

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

References

- Geldhof, G. J., Pornprasertmanit, S., Schoemann, A., & Little, T. D. (in press). Orthogonalizing through residual centering: Applications and caveats. *Educational and Psychological Measurement.*

See Also

- simFunction to use this function within a simulation study

Examples

```
dat <- residualCovariate(attitude, 2:7, 1)
```

---

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.

signature(object = "SimData") The function will random data from simData. Users may add N argument to change sample size.

signature(object = "SimMatrix") No additional arguments. The function will random parameters from simMatrix.

signature(object = "SimSet") No additional arguments. The function will random parameters from set of simMatrixs and simVectors.

signature(object = "SimMisspec") No additional arguments. The function will random parameters from set of simMatrixs and simVectors in model misspecification.

signature(object = "SimModel") The function will run an analysis specified in the SimModel object. One additional required argument is the data (put it as the second argument)

signature(object = "SimVector") No additional arguments. The function will random parameters from simVector.

signature(object = "SymMatrix") No additional arguments. The function will random parameters from symmetric simMatrix.

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
3. SimMatrix
4. SymMatrix
5. SimVector
6. SimSet
7. SimData
8. SimModel
9. SimMisspec

Examples

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

---

runFit *Build a Monte Carlo simulation that the data-generation parameters are from the result of analyzing real data*

Description

This function will analyze real data and use the result of the analysis to simulate data. Those simulated data will be analyzed by the same model. The result of the analyses will be saved in the result object, SimResult.

Usage

```
runFit(model, data, nRep=1000, misspec=new("NullSimMisspec"),
maxDraw=100, sequential=NA, facDist=new("NullSimDataDist"),
errorDist=new("NullSimDataDist"), indDist=new("NullSimDataDist"),
modelBoot=FALSE, seed=123321, silent=FALSE, multicore=FALSE,
cluster=FALSE, numProc=NULL, empiricalMissing=TRUE,
missModel=new("NullSimMissing"), usedStd=TRUE, analyzeModel=NULL)
```

Arguments

model	Model object used in analyzing the real data and simulated data (if the analyzeModel below is not specified).
data	Real data that will be used in the analysis
nRep	Number of replications.
misspec	Model *misspecification* matrices that are created by simMisspecCFA, simMisspecPath, or simMisspecSEM.
maxDraw	The maximum number of random drawn parameters and misspecification model until all parameters in the model are eligible (no negative error variance, standardized coefficients over 1).

sequential	If TRUE, use a sequential method to create data such that the data from factor are generated first and apply to a set of equations to obtain the data of indicators. If FALSE, create data directly from model-implied mean and covariance of indicators.
facDist	A SimDataDist for a distribution of factors. Use when sequential is TRUE.
errorDist	A SimDataDist for a distribution of measurement errors. Use when sequential is TRUE
indDist	A SimDataDist for a distribution of indicators. Use when sequential is FALSE.
modelBoot	Use a model-based bootstrap approach to create data. See simData for further details.
seed	Seed number
silent	TRUE if users do not wish to print number of replications during running the function.
multicore	Use multiple processors within a computer. Specify as TRUE to use it.
cluster	Not applicable now. Use for specify nodes in hpc in order to be parallelizable.
numProc	Number of processors for using multiple processors. If it is NULL, the package will find the maximum number of processors.
empiricalMissing	Use the missing pattern from the real data imposing on the simulated data if empiricalMissing=TRUE. If FALSE, the missing pattern will be generated from the missing object specified in the missModel argument. If we need a complete data, the missModel argument can be specified as blank missing object, simMissing().
missModel	A missing object used for data simulation. Only numImps attribute within the missing object is used if empiricalMissing=TRUE. All attributes are used (for generating missing data) if empiricalMissing=FALSE.
usedStd	The standardized parameters are used for data generation if usedStd=TRUE. If usedStd=FALSE, unstandardized parameters are used.
analyzeModel	The SimModel that will be used to analyze the simulated data. If NULL, the model object from the model argument is used.

Details

This function can be used to implement the Monte Carlo approach for evaluating model fit proposed by Millsap (in press). This function will use the obtained parameter estimates as the real population parameters in a simulation study, put a trivial model misspecification in the real parameters, generate multiple simulated data, analyze those simulated data, and put it in the result object, SimResult. The fit indices distribution in the result object can be used for the model fit evaluation. The useful functions for model fit evaluation are plotCutoff, getCutoff, and pValue. If modelBoot is TRUE, the function will use the Bollen-Stine bootstrap method for data generation. See simData for further details of the Bollen-Stine bootstrap approach.

Value

SimResult that saves analysis results from simulated data.

Author(s)

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

References

- Millsap, R.E. (in press). A simulation paradigm for evaluating model fit. In M. Edwards and R. MacCallum (Eds.) *Current Issues in the Theory and Application of Latent Variable Models.* New York: Routledge.

See Also

- SimModel for analysis model specification
- SimResult for the type of resulting object

Examples

```
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)
Output <- runFit(SimModel, HolzingerSwineford1939, 5, mis)
summary(Output)

out <- run(SimModel, HolzingerSwineford1939)
Output2 <- runFit(out, HolzingerSwineford1939, 5, mis)

# Bollen-Stine Bootstrap
Output3 <- runFit(out, HolzingerSwineford1939, 5, modelBoot=TRUE)

# Bollen-Stine Bootstrap with trivial misspecification
Output4 <- runFit(out, HolzingerSwineford1939, 5, mis, modelBoot=TRUE)

# Example with multiple imputation
library(lavaan)
loading <- matrix(0, 11, 3)
loading[1:3, 1] <- NA
loading[4:7, 2] <- NA
loading[8:11, 3] <- NA
path <- matrix(0, 3, 3)
path[2:3, 1] <- NA
path[3, 2] <- NA
errorCov <- diag(NA, 11)
facCov <- diag(3)
param <- simParamSEM(LY=loading, BE=path, TE=errorCov, PS=facCov)

miss <- simMissing(pmMCAR=0.03, numImps=5)
usedData <- run(miss, PoliticalDemocracy)

model <- simModel(param, indLab=c(paste("x", 1:3, sep=""), paste("y", 1:8, sep="")))
out <- run(model, usedData, miss)
output <- runFit(model, usedData, 5, missModel=miss)
pValue(out, output)
```

`runFitParam` *Build a parameter result object that the data-generation parameters are from the result of analyzing real data*

Description

This function will analyze real data and use the result of the analysis to find the range of parameters used in data simulation.

Usage

```
runFitParam(model, nRep = 1000, misspec = new("NullSimMisspec"),
maxDraw = 100, seed = 123321, usedStd = TRUE, ...)
```

Arguments

`model`	Model object used in analyzing the real and simulated data.
`nRep`	Number of replications.
`misspec`	Model *misspecification* matrices that are created by `simMisspecCFA`, `simMisspecPath`, or `simMisspecSEM`.
`maxDraw`	The maximum number of random drawn parameters and misspecification model until all parameters in the model are eligible (no negative error variance, standardized coefficients over 1).
`seed`	Seed number
`usedStd`	The standardized parameters are used for data generation if `usedStd=TRUE`. If `usedStd=FALSE`, unstandardized parameters are used.
`...`	The only additional argument is `data`, which is the real data that will be used in the analysis. The data argument is used when the `model` is `linkS4class{SimModel}`

Details

This function will use the obtained parameter estimates as the real population parameters in a simulation study, put a trivial model misspecification in the real parameters, and report the real and misspecified parameters in the `SimResultParam` object.

Value

`SimResultParam` that saves the parameters and misspecification used in each replication.

Author(s)

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

See Also

- `SimModel` for analysis model specification
- `SimResult` for the type of resulting object
- `runFit` for data simulation based on real data

Examples

```
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)
Output <- runFitParam(SimModel, data=HolzingerSwineford1939, nRep=5, misspec=mis)
summary(Output)

out <- run(SimModel, HolzingerSwineford1939)
Output2 <- runFitParam(out, nRep=5, misspec=mis)
```

---

runLavaan *Run data by the model object by the* lavaan *package*

Description

Transform model object (SimModel) to lavaan script, run the obtained data, and make model output object (SimModelOut)

Usage

```
runLavaan(object, Data, miss="fiml", estimator="ML")
```

Arguments

object	The model object (SimModel)
Data	The data to be analyzed
miss	The method to handle missing data in lavaan. The default is full information maximum likelihood.
estimator	The method of estimation. The default is maximum likelihood (ML).

Value

The model output object that saves parameter estimates, standard errors, and fit indices.

Author(s)

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

Examples

```
# No example
```

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)

  }
```

---

runMisspec *Draw actual parameters and model misspecification*

Description

Randomly draw actual parameters from SimSet object and randomly draw the misspecified parameters from SimMisspec object. Then, the function will put the misspecified parameters on top of the actual parameters.

Usage

```
runMisspec(object, misspec, SimEqualCon=new("NullSimEqualCon"))
```

Arguments

object	The SimSet object that saves the actual parameters
misspec	The SimSet object that saves the misspecified parameters
SimEqualCon	The equality constraint to equate parameters

Details

When the equality constraint does not exist, there are two orders of generating parameters:

1. SimMisspec@misBeforeFill=TRUE The misspecification will be added into user-specified parameters and the auto-completion on the other parameters.

2. `SimMisspec@misBeforeFill=FALSE` The auto-completion will be applied first, then the misspecification will be added, and finally the auto-completion works again to adjust the parameter after adding misspecification.

When the equality constraint exists, there are six orders of generating parameters:

1. `SimMisspec@misBeforeFill=TRUE & SimMisspec@conBeforeMis=TRUE & SimEqualCon@conBeforeFill=TRU` The equality constraint is applied first, then the misspecification is added, and finally the auto-completion is applied.
2. `SimMisspec@misBeforeFill=FALSE & SimMisspec@conBeforeMis=TRUE & SimEqualCon@conBeforeFill=TR` The equality constarint is applied first, then the auto-completion is applied, the misspecification will be added, and finally the auto-completion is used to adjusted for the added parameters.
3. `SimMisspec@misBeforeFill=FALSE & SimMisspec@conBeforeMis=TRUE & SimEqualCon@conBeforeFill=FA` The auto-completeion is applied first, then the equality constriant is applied with adjusting parameters by the auto-completion, and finally the misspecification is added with adjusting the parameters by the auto-completion.
4. `SimMisspec@misBeforeFill=FALSE & SimMisspec@conBeforeMis=FALSE & SimEqualCon@conBeforeFill=F` The auto-completeion is applied first, then the misspecification is added with adjusting paramters by the auto-completion, and finally the equality constraint is added with adjusting parameters by the auto-completion.
5. `SimMisspec@misBeforeFill=TRUE & SimMisspec@conBeforeMis=FALSE & SimEqualCon@conBeforeFill=TR` The misspecification is added first, then the equality constraint is applied, and finally the auto-completion is applied.
6. `SimMisspec@misBeforeFill=TRUE & SimMisspec@conBeforeMis=FALSE & SimEqualCon@conBeforeFill=FA` The misspecification is added first, then the auto-completion is applied, and finally the equality constriant is applied with adjusting parameters by the auto-completion.

The default in this program is the first option for both scenarios. Other scenarios will be useful when users would like to put model misspecification or equality constraints on the parameters that users do not manually specify.

Value

A list of with two elements: 1) `param` `SimRSet` of real model parameters, 2) `misspec` `SimRSet` of real model parameters with model misspecification, and 3) `misspecAdd` the misspecification alone

Author(s)

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

Examples

```
# No example
```

runRep *Run one replication within a big simulation study*

Description

Run one replication within a big simulation study within the simResult function.

Usage

```
runRep(object, objData, objModel, objMissing=new("NullSimMissing"), objFunction=new("NullSimFunc
```

Arguments

object	A list of varying parts across replications: 1) list of real parameters and model misspecification, 2) sample size, 3) percent missing completely at random, 4) percent missing at random, 5) seed number
objData	The data object, SimData, used in the simulation study
objModel	The model object, SimModel, used in the simulation study
objMissing	The missing object, SimMissing, used in the simulation study
objFunction	The function object, SimFunction, used in the simulation study
silent	If TRUE, no warning or printout as much as possible

Value

A list that contains

- coef parameter estimates
- se standard errors
- fit Model fit indices
- converged Converged?
- param Parameter values provided from model output object
- FMI1 Fraction missing method 1
- FMI2 Fraction missing method 2
- std Standardized coeffcient
- paramData Parameter underlying generated data

Author(s)

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

Examples

```
# No example
```

setOpenMxObject *Rearrange starting values for* OpenMx

Description

Rearrange starting values such that it is appropriate for OpenMx matrix specification such that free parameters are set to be TRUE/FALSE and values meaning be both fixed value or starting values

Usage

```
setOpenMxObject(param, start)
```

Arguments

param	Set of free parameters (NA = free parameters; numbers = fixed value with a specified number)
start	Parameter/Starting values of all free parameters

Value

A vector, a matrix, or a SimRSet which includes numbers of fixed parameters and starting values of free parameters.

Methods

signature(param="vector", start="vector") This function will add fixed value from parameters vector into starting value vector

signature(param="matrix", start="matrix") This function will add fixed value from parameters matrix into starting value matrix

signature(param="SimParam", start="SimRSet") This function will put all fixed values in SimParam into set of starting value matrices, SimRSet

Author(s)

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

Examples

```
# This function is not public

# parameter <- c(NA, NA, 0, 0)
# startingValues <- c(2, 5, 0, 0)
# setOpenMxObject(parameter, startingValues)
```

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 can be linkS4class{SimModelOut} or 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

- SimModelOut for model output object
- SimResult for result object

Examples

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

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)
```

simData *Create a Data object*

Description

This function will be used to create data specification and ready for data simulation.

Usage

```
simData(param, ...)
```

Arguments

param	Model specification matrices that are created by simSetCFA, simSetPath, or simSetSEM or the analysis output.
...	Other values that will be explained specifically for each class

Details

This function will use mvrnorm function in MASS package to create data from model implied covariance matrix if the data distribution object (SimDataDist) is not specified. It the data distribution object is specified, the Gaussian copula model is used. See SimDataDist for further details. For the model-based bootstrap, the transformation proposed by Yung & Bentler (1996) is used. This procedure is the expansion from the Bollen and Stine (1992) bootstrap including a mean structure. The model-implied mean vector and covariance matrix with trivial misspecification will be used in the model-based bootstrap if misspec is specified. See page 133 of Bollen and Stine (1992) for a reference.

Value

SimData object that save data model specification.

Details in ...

- *n:* Desired sample size
- *misspec:* Model *misspecification* matrices that are created by simMisspecCFA, simMisspecPath, or simMisspecSEM.
- *equalCon:* Equality constraints that are created by simEqualCon. This will specify equality econstraints of parameters in data generation process.
- *maxDraw:* The maximum number of random drawn parameters and misspecification model until all parameters in the model are eligible (no negative error variance, standardized coefficients over 1).
- *sequential:* If TRUE, use a sequential method to create data such that the data from factor are generated first and apply to a set of equations to obtain the data of indicators. If FALSE, create data directly from model-implied mean and covariance of indicators.
- *facDist:* A SimDataDist for a distribution of factors. Use when sequential is TRUE.
- *errorDist:* A SimDataDist for a distribution of measurement errors. Use when sequential is TRUE
- *indDist:* A SimDataDist for a distribution of indicators. Use when sequential is FALSE.

- *indLab:* A vector of indicator names. If not specified, the variables names are y1, y2, ...
- *modelBoot:* If TRUE, use a model-based bootstrap for data generation. See details for further information. This argument need a dataset in the realData argument.
- *realData:* The real dataset that the model based bootstrap will follow the distribution.
- *usedStd:* If SimModelOut is used for data generation, standardized parameters are used for data generation if usedStd=TRUE. If usedStd=FALSE, unstandardized parameters are used.

Author(s)

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

References

- Bollen, K. A., & Stine, R. A. (1992). Bootstrapping goodness-of-fit measures in structural equation models. *Sociological Methods and Research, 21,* 205-229.
- Yung, Y.-F., & Bentler, P. M. (1996). Bootstrapping techniques in analysis of mean and covariance structures. In G. A. Marcoulides & R. E. Schumacker (Eds.), *Advanced structural equation modeling: Issues and techniques* (pp. 195-226). Mahwah, NJ: Erlbaum.

See Also

- simSetCFA to see CFA model specification
- simSetPath to see Path analysis model specification
- simSetSEM to see SEM model specification
- simMisspecCFA for specifying misspecification in CFA model
- simMisspecPath for specifying misspecification in Path analysis model
- simMisspecSEM for specifying misspecification in SEM model
- simEqualCon for setting equality constraints.
- simDataDist for data distribution object set-up for facDist, errorDist, or indDist arguments.

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
LX <- simMatrix(loading, loadingValues)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, 0.5)
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTD <- symMatrix(error.cor)
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 200)
summary(SimData)
run(SimData)

# With Misspecification Model
```

```
n01 <- simNorm(0, 0.1)
error.cor.Mis <- matrix(NA, 6, 6)
diag(error.cor.Mis) <- 1
RTD.Mis <- symMatrix(error.cor.Mis, "n01")
CFA.Model.Mis <- simMisspecCFA(RTD=RTD.Mis)
SimData <- simData(CFA.Model, 200, misspec=CFA.Model.Mis)
summary(SimData)
run(SimData)
```

SimData-class *Class* "SimData"

Description

This class will save information for data simulation and can create data by run function.

Objects from the Class

Objects can be created by simData. Also, it can be called by new("SimData", ...).

Slots

modelType: Model type (CFA, Path, or SEM)

n: Sample size

param: Model specification that used in data generation. It must be in SimSet class.

misspec: Model misspecification that used in data generation. It must be in SimMisspec class.

equalCon: Equality constraints in data generation. It must be in SimEqualCon class.

maxDraw: The maximum number of random drawn parameters and misspecification model until all parameters in the model are eligible (no negative error variance, standardized coefficients over 1).

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

facDist: A SimDataDist for a distribution of factors. Use when sequential is TRUE.

errorDist: A SimDataDist for a distribution of measurement errors. Use when sequential is TRUE.

indDist: A SimDataDist for a distribution of indicators. Use when sequential is FALSE.

indLab: A vector of indicator names. If not specified, the variables names are y1, y2, ...

modelBoot: If TRUE, use a model-based bootstrap for data generation. See details for further information. This argument need a dataset in the realData argument.

realData: The real dataset that the model based bootstrap will follow the distribution.

Methods

run To create data from this class. N is the additional argument that users may change the sample size when creating data. dataOnly is default to be TRUE. If FALSE, the resulting object in SimDataOut can be used to provide details of things used in create the data.

summary Summarize all attributes in the simData.

Author(s)

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

See Also

- linkS4class{SimSet} for how to specify data generation model.
- linkS4class{SimMisspec} for how to specify misspecification in this data generation model.
- linkS4class{SimEqualCon} for how to set equality constraints for data generation.
- link{simResult} for the use of this class to run Monte Carlo simulation.
- linkS4class{SimModelOut} for the output type after the run function with dataOnly=TRUE.

Examples

```
showClass("SimData")
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 <- simMatrix(loading, loadingValues)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, 0.5)
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTD <- symMatrix(error.cor)
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 200)
summary(SimData)
run(SimData)
```

---

simDataDist *Create a data distribution object.*

Description

This function will create simResult by different ways. One way is to create data and analyze data multiple times by specifying SimData and SimModel and save it in the SimDataDist.

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

- SimData for data model specification
- SimModel for analysis model specification
- SimResult for the type of resulting object

Examples

```
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- simMatrix(loading, 0.7)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 500)
SimModel <- simModel(CFA.Model)
# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output <- simResult(5, SimData, SimModel)
#summary(Output)
```

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

```
showClass("SimDataDist")

chisq3 <- simChisq(3)
chisq8 <- simChisq(8)
dist <- simDataDist(chisq3, chisq8)
dist2 <- extract(dist, 2)

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)
```

SimDataOut-class *Class* "SimDataOut"

Description

This class will provide the simulated dataset and population behind the generated dataset.

Objects from the Class

Objects can be created by run on the SimData with dataOnly=FALSE. It can also be called from the form new("SimDataOut", ...).

Slots

modelType: Analysis model type (CFA, Path, or SEM)

data: The simulated data

param: Model specification that used in data generation. It must be in SimSet class.

paramOut: Parameter values underlying the simulated data.

misspecOut: Model misspecification underlying the simulated data

equalCon: Equality constraints in data generation. It must be in SimEqualCon class.

n: Sample size of the created data.

Methods

- summary to summarize the object
- createImpliedMACS to create the model implied means and covariance matrix from the parameter estimates. The misspec argument can be specified as TRUE to create the model implied means and covariance matrix from both parameters and model misspecification
- summaryPopulation to summarize the data generation population underlying the data.
- getPopulation to extract the data generation population. The additional argument is misspec. If TRUE, the data generation population with model misspecification will be returned. If FALSE, the data generation population without model misspecification will be returned.

Author(s)

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

See Also

- SimData The object used as data template for simulated data.
- SimModel The object used as analysis

Examples

```
showClass("SimDataOut")
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- simMatrix(loading, 0.7)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 500)
SimModel <- simModel(CFA.Model)
Data <- run(SimData, dataOnly=FALSE)
Result <- run(SimModel, Data)
summary(Data)
summaryPopulation(Data)
mis <- getPopulation(Data, misspec=TRUE)
```

---

simEqualCon *Equality Constraint Object*

Description

This function will be used to specify equality constraints.

Usage

```
simEqualCon(..., modelType, conBeforeFill=TRUE)
```

Arguments

`...`	Each equality constraint in the model will be specified as a matrix. Rows represent elements that users wish to constrain. For single-group analysis, two columns are needed in the matrix. The first column indicates row of elements and second columns indicates columns of elements. Rownames will represent the matrix of elements that they are in. The detail section will discuss about how to specify row names. The first example shown below will show how to specify equality constraints for `LY(1,1)`, `LY(2,1)`, and `LY(3,1)`. For multiple groups, the columns will be three instead. The first column represent groups. The second and third columns represent row and column, respectively. The second example shown below will show how to specify equality constraints for `BE(2, 1)` of two groups. If you have multiple equality constriants, you can make multiple matrices to represent them and add in the function. See the third example for multiple constraints.
`modelType`	Type of analysis: `CFA`, `Path`, `Path.exo`, `SEM`, or `SEM.exo`.
`conBeforeFill`	`TRUE` if users wish to apply equality constraint before applying the auto-completion on the parameters that users have not specified. `FALSE` if users wish to apply the auto-completion before applying equality constraint. This option is helpful when users wish to apply the equality constraint on the parameters that users have not specified (e.g., constraining the residual variance, which users let the package to calculate it and not specify it). See `runMisspec` for further details.

Details

Row names specification depends on type of model. If users specify CFA model, the specification in shown in simSetCFA function. If users specify Path analysis with or without exogenous variables, the specification is shown in simSetPath function. If users specify SEM model with or without exogenous variables, the specification is shown in simSetSEM function. However, basically, the names of matrices you put in these function are also eligible for this function as well.

Value

Object in SimEqualCon that save those equality constraints.

Note

The available constraints now are equality constraints. We expect to create nonlinear constraints soon.

Author(s)

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

See Also

- simSetCFA to see model specification in CFA model
- simSetPath to see model specification in Path analysis model
- simSetSEM to see model specification in SEM model
- SimEqualCon for the simResult

Examples

```
# Example 1: Single-group, one constraint
constraint <- matrix(0, 3, 2)
constraint[1,] <- c(1, 1)
constraint[2,] <- c(2, 1)
constraint[3,] <- c(3, 1)
rownames(constraint) <- rep("LY", 3)
equal.loading <- simEqualCon(constraint, modelType="SEM.exo")

# Example 2: Multiple-group, one constraint
group.con <- matrix(0, 2, 3)
group.con[1,] <- c(1, 2, 1)
group.con[2,] <- c(2, 2, 1)
rownames(group.con) <- rep("BE", 2)
equal.path <- simEqualCon(group.con, modelType="Path")

# Example 3: Single-group, multiple constraints
constraint1 <- matrix(1, 3, 2)
constraint1[,1] <- 1:3
rownames(constraint1) <- rep("LY", 3)
constraint2 <- matrix(2, 3, 2)
constraint2[,1] <- 4:6
rownames(constraint2) <- rep("LY", 3)
constraint3 <- matrix(3, 2, 2)
constraint3[,1] <- 7:8
rownames(constraint3) <- rep("LY", 2)
```

```
equal.loading2 <- simEqualCon(constraint1, constraint2, constraint3, modelType="SEM")
summary(equal.loading2)
```

SimEqualCon-class *Class* "SimEqualCon"

Description

Set of specified equality constraints

Details

The Equality slot contains list of equality constraint. Each element in the list is an individual equality constraint saved in a matrix. Each row represents each element. If the matrix has two columns, the first column indicates row of the element and the second column indicates column of the element. If the matrix has three columns, the first column is the group of matrix. The rest is row and column. Row name represents the matrix that the element is in. The definition of row name can be seen in simSetCFA, simSetPath, or simSetSEM, depending on analysis model you specify.

Objects from the Class

Objects can be created by simEqualCon. Also, it can be called of the form new("SimEqualCon", ...).

Slots

con: List of equality constraint. See the Details section for the description of each equality constraint.

modelType: Analysis model (CFA, SEM, Path)

conBeforeFill: TRUE if users wish to apply equality constraint before applying the auto-completion on the parameters that users have not specified. FALSE if users wish to apply the auto-completion before applying equality constraint.

Methods

summary Summarize all attributes of this object

Author(s)

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

See Also

- simEqualCon for the constructor of this class
- simData for a potential use of this object to create data
- simModel for a potential use of this object to run an analysis

Examples

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

---

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 simResult)

Author(s)

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

See Also

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

Examples

```
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)

fun <- simFunction(indProd, 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 simResult.

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

```
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)

fun <- simFunction(indProd, 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(indProd, var1=1:3, var2=4:6)
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)
```

SimGenLabels-class *Class* "SimGenLabels"

Description

The label of parameter values in the data generation parameterization format.

Objects from the Class

Object can be created by makeLabels, Objects can be also created by calls of the form new("SimGenLabels", ...).

Slots

modelType: Model type (CFA, Path, or SEM)

LY: The labels of the factor loading matrix between endogenous factors and Y indicators

TE: The labels of the covariance matrix between Y measurement error

RTE: The labels of the correlation matrix between Y measurement error

VTE: The labels of the variance of Y measurement error

PS: The labels of the residual covariance of endogenous factors

RPS: The labels of the residual correlation of endogenous factors

VPS: The labels of the residual variances of endogenous factors

BE: The labels of the regression effect among endogenous factors

TY: The labels of the measurement intercepts of Y indicators

AL: The labels of the factor intercepts of endogenous factors

ME: The labels of the factor means of endogenous factors

MY: The labels of the total Mean of Y indicators

VE: The labels of the total variance of endogenous factors

VY: The labels of the total variance of Y indicators

LX: The labels of the factor loading matrix between exogenous factors and X indicators

TD: The labels of the covariance matrix between X measurement error

RTD: The labels of the correlation matrix between X measurement error

VTD: The labels of the variance of X measurement error

PH: The labels of the covariance among exogenous factors

RPH: The labels of the correlation among exogenous factors

GA: The labels of the regreeion effect from exogenous factors to endogenous factors

TX: The labels of the measurement intercepts of X indicators

KA: The labels of the factor Mean of exogenous factors

MX: The labels of the total Mean of X indicators

VPH: The labels of the variance of exogenous factors

VX: The labels of the total variance of X indicators

TH: The labels of the measurement error covariance between X indicators and Y indicators

RTH: The labels of the measurement error correlation between X indicators and Y indicators

Extends

Class "MatrixSet", directly.

Methods

summary Get the summary of model specification

Author(s)

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

See Also

- MatrixSet

Examples

```
# No example
```

---

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` *Create simMatrix that save free parameters and starting values, as well as fixed values*

Description

Create `SimMatrix` object that save free parameters and starting values, as well as fixed values. This will be used for model specification later, such as for factor loading matrix or regression coefficient matrix.

Usage

```
simMatrix(free = NULL, value = NULL)
```

Arguments

`free`	Matrix of free parameters. Use `NA` to specify free parameters. Use number as fixed value (including zero). If this argument is not specified, the information from the `value` argument is used. The positions in the `value` argument that are 0 or "" are fixed parameters as 0. The other positions are free parameters.
`value`	Starting values. Can be either one element or matrix with the same dimension as free parameter matrix. Each element can be numbers (in either `as.numeric` or `as.character` format) or the name of distribution object `VirtualDist`.

Value

`SimMatrix` object that will be used for model specification later.

Author(s)

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

See Also

- See `VirtualDist` for the resulting object.
- See `symMatrix` for creating symmetric simMatrix.
- See `simVector` for simVector.

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
LX <- simMatrix(loading, loadingValues)
summary(LX)
run(LX)

n65 <- simNorm(0.6, 0.05)
LY <- simMatrix(loading, "n65")
summary(LY)
run(LY)

start <- matrix(0, 6, 2)
start[1:3, 1] <- 0.7
start[4:6, 2] <- 0.7
ST <- simMatrix(value=start)
```

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 "simMatrix" function. Objects can be also created by calls of the form new("SimMatrix", ...).

Slots

free: indicates which elements of the matrix are free or fixed. "NA" means the element is freely estimated. Numbers (including 0) means the element is fixed to be the indicated number.

value: indicates the starting values of each element in the matrix. The starting values could be numbers or the name of "distribution objects"

Methods

adjust Adjust an element in the "SimMatrix" object

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

extract Extract elements from a simMatrix. The additional arguments are the indicies of rows and columns to be extracted.

Author(s)

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

See Also

- SymMatrix for symmetric random parameter matrix
- 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 <- simMatrix(loading, loadingValues)
summary(LX)
run(LX)

n65 <- simNorm(0.6, 0.05)
LY <- simMatrix(loading, "n65")
summary(LY)
run(LY)

u34 <- simUnif(0.3, 0.4)
LY <- adjust(LY, "u34", c(2, 1))
summary(LY)
run(LY)
summaryShort(LY)

LY <- extract(LY, 1:3, 1)
summary(LY)
```

---

simMissing *Construct a SimMissing object to create data with missingness and analyze missing data.*

Description

Function creates a SimMissing object that can be passed to simResult for creating and analyzing data with missingness.

Usage

```
simMissing(cov=0, pmMCAR=0, pmMAR=0, nforms=0, itemGroups=list(0), timePoints=1, twoMethod=0, pr
```

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)
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.
timePoints	Number of timepoints items were measured over. For longitudinal data, planned missing designs will be implemented within each timepoint.
numImps	The number of imputations to be used when multiply imputing missing data. Setting numImps to 0 will use FIML to handle missing data.
ignoreCols	The columns not imposed any missing values for any missing data patterns
threshold	The threshold of covariates that divide between the area to impose missing and the area not to impose missing. The default threshold is the mean of the covariate.
covAsAux	If TRUE, the covariate listed in the object will be used as auxiliary variables when putting in the model object. If FALSE, the covariate will be included in the analysis.
logical	A matrix of logical values (TRUE/FALSE). If a value in the dataset is corresponding to the TRUE in the logical matrix, the value will be missing.
...	Additional arguments to be passed to amelia for imputation. Only used if numImps is greater than 0.

Details

Without specifying any any arguments, no missingness will be introduced. Covariates are required to impose MAR missing. Imputations will be performed with Amelia

Value

A simMissing object to be used with SimResult.

Author(s)

Alexander M. Schoemann (University of Kansas; <schoemann@ku.edu>) Patrick Miller(University of Kansas; <patr1ckm@ku.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 simResult function.
- runMI for imputing missing data by multiple imputation and analyze the imputed data.

Examples

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

loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- simMatrix(loading, 0.7)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 500)
SimModel <- simModel(CFA.Model)

#Create data
dat <- run(SimData)

#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 <- simMissing(nforms=3, timePoints=3, numImps=10)
```

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.

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

numImps: The number of imputations to be used when multiply imputing missing data. Setting numImps to 0 will use FIML to handle missing data.

impMethod: Package that will be used for imputation. Currently only Amelia is supported.

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.

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.

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.

opts: A list of additional options to be passed to the imputation method specified in impMethod.

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
```

SimMisspec-class *Class* "SimMisspec"

Description

Misspecification model added on true model specifiction. This class contains SimVector, SimMatrix, and SymMatrix specifying misspecification.

Objects from the Class

Object can be created by simMisspecCFA, simMisspecPath, or simMisspecSEM, for CFA, Path analysis, or SEM model, respectively. Objects can be also created by calls of the form new("SimMisspec", ...).

Slots

`modelType`: Model type (CFA, Path, or SEM)

`LY`: Factor loading matrix between endogenous factors and Y indicators

`TE`: Covariance matrix between Y measurement error

`RTE`: Correlation matrix between Y measurement error

`VTE`: Variance of Y measurement error

`PS`: Residual covariance of endogenous factors

`RPS`: Residual correlation of endogenous factors

`VPS`: Residual variances of endogenous factors

`BE`: Regression effect among endogenous factors

`TY`: Measurement intercepts of Y indicators

`AL`: Factor intercepts of endogenous factors

`ME`: Factor means of endogenous factors

`MY`: Total Mean of Y indicators

`VE`: Total variance of endogenous factors

`VY`: Total variance of Y indicators

`LX`: Factor loading matrix between exogenous factors and X indicators

`TD`: Covariance matrix between X measurement error

`RTD`: Correlation matrix between X measurement error

`VTD`: Variance of X measurement error

`PH`: Covariance among exogenous factors

`RPH`: Correlation among exogenous factors

`GA`: Regreeion effect from exogenous factors to endogenous factors

`TX`: Measurement intercepts of X indicators

`KA`: Factor Mean of exogenous factors

`MX`: Total Mean of X indicators

`VPH`: Variance of exogenous factors

`VX`: Total variance of X indicators

`TH`: Measurement error covariance between X indicators and Y indicators

`RTH`: Measurement error correlation between X indicators and Y indicators

`conBeforeMis`: `TRUE` if users wish to constrain parameters before adding misspecification. `FALSE` if users wish to constrain parameters after adding misspecification.

`misBeforeFill`: `TRUE` if users wish to apply misspecification before applying the auto-completion on the parameters that users have not specified. `FALSE` if users wish to apply the auto-completion before adding misspecification. This option is helpful when users wish to apply misspecification on the parameters that users have not specified (e.g., adding trivial misspecification on the residual variance, which users let the package to calculate it and not specify it). See `runMisspec` for further details.

`misfitType`: The type of population misfit used in the `misfitBound` below. The default is `"rmsea"`. The two other options are `"f0"` and `"srmr"`. See `popMisfitMACS` for further details.

`misfitBound`: The lower and upper bounds of the population misfit. This option must be a vector with two elements.

`averageNumMisspec`: If `TRUE`, the misfit will be divided by the number of free elements in the misspecification object. The default is `FALSE`.

`optMisfit`: Use the optimization method to pick the misspecification set. That is, the program will draw a number of misspecification sets. Then, the different sets of misspecification will be compared together. If `"min"` is specified, the program will pick the misspecification set the provides the least amount of misfit. If `"max"` is specified, the program will pick the set that has the largets misfit. The default is `"none"` to not use the optimization method.

`numIter`: The number of different misspecification sets for comparison in the optimization method.

Extends

Class "SimSet", directly.

Methods

summary Provide the brief description of this object.

run Create a sample of parameters in this object. In other words, draw a sample from all random parameters which is represented in `VirtualDist`.

Author(s)

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

See Also

- Create an object this class by CFA, Path Analysis, or SEM model by `simMisspecCFA`, `simMisspecPath`, or `simMisspecSEM`, respectively.
- See how to specify true model by `SimSet`.

Examples

```
showClass("SimMisspec")
n01 <- simNorm(0, 0.1)
error.cor.Mis <- matrix(NA, 6, 6)
diag(error.cor.Mis) <- 1
RTD.Mis <- symMatrix(error.cor.Mis, "n01")
CFA.Model.Mis <- simMisspecCFA(RTD=RTD.Mis)
```

---

simMisspecCFA *Set of model misspecification for CFA model.*

Description

This function will define model misspecifiction from a defined model. This function is similar to `simSetCFA` such that the matrices that indicates misspecification will be added as arguments in the function. However, users do not have to add all matrices and vectors in the function. Only element indicating misspecification is added.

Usage

```
simMisspecCFA(..., conBeforeMis=TRUE, misBeforeFill=TRUE,
misfitType="rmsea", misfitBound=new("NullVector"), averageNumMisspec=FALSE,
optMisfit="none", numIter=20)
```

Arguments

`...`	Arguments definition is listed in the `Details` section of `simSetCFA`. Again, this function does not require to list all required matrices or vectors like the `simSetCFA` function. Only misspecification is added.
`conBeforeMis`	`TRUE` if users wish to constrain parameters before adding misspecification. `FALSE` if users wish to constrain parameters after adding misspecification.
`misBeforeFill`	`TRUE` if users wish to apply misspecification before applying the auto-completion on the parameters that users have not specified. `FALSE` if users wish to apply the auto-completion before adding misspecification. This option is helpful when users wish to apply misspecification on the parameters that users have not specified (e.g., adding trivial misspecification on the residual variance, which users let the package to calculate it and not specify it). See `runMisspec` for further details.
`misfitType`	The type of population misfit used in the `misfitBound` below. The default is `"rmsea"`. The two other options are `"f0"` and `"srmr"`. See `popMisfitMACS` for further details.
`misfitBound`	The lower and upper bounds of the population misfit. This option must be a vector with two elements.
`averageNumMisspec`	If `TRUE`, the misfit will be divided by the number of free elements in the misspecification object. The default is `FALSE`.
`optMisfit`	Use the optimization method to pick the misspecification set. That is, the program will draw a number of misspecification sets. Then, the different sets of misspecification will be compared together. If `"min"` is specified, the program will pick the misspecification set the provides the least amount of misfit. If `"max"` is specified, the program will pick the set that has the largets misfit. The default is `"none"` to not use the optimization method.
`numIter`	The number of different misspecification sets for comparison in the optimization method.

Value

object in `SimMisspec` that saves model misspecification.

Author(s)

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

See Also

- `simSetCFA` for matrix definition and how to specify CFA model
- `SimMisspec` for the simResult
- `simMisspecPath` for misspecification model in Path analysis and `simMisspecSEM` for misspecification model in SEM.

Examples

```
n01 <- simNorm(0, 0.1)
error.cor.Mis <- matrix(NA, 6, 6)
diag(error.cor.Mis) <- 1
RTD.Mis <- symMatrix(error.cor.Mis, "n01")
CFA.Model.Mis <- simMisspecCFA(RTD=RTD.Mis)
```

---

simMisspecPath *Set of model misspecification for Path analysis model.*

Description

This function will define model misspecifiction from a defined model. This function is similar to simSetPath such that the matrices that indicates misspecification will be added as arguments in the function. However, users do not have to add all matrices and vectors in the function. Only element indicating misspecification is added.

Usage

```
simMisspecPath(..., exo = FALSE, conBeforeMis=TRUE, misBeforeFill=TRUE,
misfitType="rmsea", misfitBound=new("NullVector"), averageNumMisspec=FALSE,
optMisfit="none", numIter=20)
```

Arguments

...	Arguments definition is listed in the Details section of simSetPath. Again, this function does not require to list all required matrices or vectors like the simSetPath function. Only misspecification is added.
exo	specify TRUE if users wish to specify both exogenous and endogenous indicators.
conBeforeMis	TRUE if users wish to constrain parameters before adding misspecification. FALSE if users wish to constrain parameters after adding misspecification.
misBeforeFill	TRUE if users wish to apply misspecification before applying the auto-completion on the parameters that users have not specified. FALSE if users wish to apply the auto-completion before adding misspecification. This option is helpful when users wish to apply misspecification on the parameters that users have not specified (e.g., adding trivial misspecification on the residual variance, which users let the package to calculate it and not specify it). See runMisspec for further details.
misfitType	The type of population misfit used in the misfitBound below. The default is "rmsea". The two other options are "f0" and "srmr". See popMisfitMACS for further details.
misfitBound	The lower and upper bounds of the population misfit. This option must be a vector with two elements.
averageNumMisspec	If TRUE, the misfit will be divided by the number of free elements in the misspecification object. The default is FALSE.

optMisfit — Use the optimization method to pick the misspecification set. That is, the program will draw a number of misspecification sets. Then, the different sets of misspecification will be compared together. If "min" is specified, the program will pick the misspecification set the provides the least amount of misfit. If "max" is specified, the program will pick the set that has the largets misfit. The default is "none" to not use the optimization method.

numIter — The number of different misspecification sets for comparison in the optimization method.

Value

object in SimMisspec that saves model misspecification.

Author(s)

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

See Also

- simSetPath for matrix definition and how to specify Path analysis model
- SimMisspec for the simResult
- simMisspecCFA for misspecification model in CFA and simMisspecSEM for misspecification model in SEM.

Examples

```
u1 <- simUnif(-0.1, 0.1)
mis.path.GA <- matrix(0, 2, 2)
mis.path.GA[2, 1:2] <- NA
mis.GA <- simMatrix(mis.path.GA, "u1")
Path.Mis.Model <- simMisspecPath(GA = mis.GA, exo=TRUE)
```

---

simMisspecSEM *Set of model misspecification for SEM model.*

Description

This function will define model misspecifiction from a defined model. This function is similar to simSetSEM such that the matrices that indicates misspecification will be added as arguments in the function. However, users do not have to add all matrices and vectors in the function. Only element indicating misspecification is added.

Usage

```
simMisspecSEM(..., exo = FALSE, conBeforeMis=TRUE, misBeforeFill=TRUE,
misfitType="rmsea", misfitBound=new("NullVector"), averageNumMisspec=FALSE,
optMisfit="none", numIter=20)
```

Arguments

...	Arguments definition is listed in the Details section of simSetSEM. Again, this function does not require to list all required matrices or vectors like the simSetSEM function. Only misspecification is added.
exo	specify TRUE if users wish to specify both exogenous and endogenous indicators.
conBeforeMis	TRUE if users wish to constrain parameters before adding misspecification. FALSE if users wish to constrain parameters after adding misspecification.
misBeforeFill	TRUE if users wish to apply misspecification before applying the auto-completion on the parameters that users have not specified. FALSE if users wish to apply the auto-completion before adding misspecification. This option is helpful when users wish to apply misspecification on the parameters that users have not specified (e.g., adding trivial misspecification on the residual variance, which users let the package to calculate it and not specify it). See runMisspec for further details.
misfitType	The type of population misfit used in the misfitBound below. The default is "rmsea". The two other options are "f0" and "srmr". See popMisfitMACS for further details.
misfitBound	The lower and upper bounds of the population misfit. This option must be a vector with two elements.
averageNumMisspec	If TRUE, the misfit will be divided by the number of free elements in the misspecification object. The default is FALSE.
optMisfit	Use the optimization method to pick the misspecification set. That is, the program will draw a number of misspecification sets. Then, the different sets of misspecification will be compared together. If "min" is specified, the program will pick the misspecification set the provides the least amount of misfit. If "max" is specified, the program will pick the set that has the largets misfit. The default is "none" to not use the optimization method.
numIter	The number of different misspecification sets for comparison in the optimization method.

Value

object in SimMisspec that saves model misspecification.

Author(s)

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

See Also

- simSetSEM for matrix definition and how to specify SEM model
- SimMisspec for the simResult
- simMisspecCFA for misspecification model in CFA and simMisspecPath for misspecification model in Path analysis.

Examples

```
u2 <- simUnif(-0.2, 0.2)
n1 <- simNorm(0, 0.1)
loading.X.trivial <- matrix(NA, 6, 2)
loading.X.trivial[is.na(loading.X.trivial)] <- 0
LX.trivial <- simMatrix(loading.X.trivial, "u2")
error.cor.X.trivial <- matrix(NA, 6, 6)
diag(error.cor.X.trivial) <- 0
RTD.trivial <- symMatrix(error.cor.X.trivial, "n1")
error.cor.Y.trivial <- matrix(NA, 2, 2)
diag(error.cor.Y.trivial) <- 0
RTE.trivial <- symMatrix(error.cor.Y.trivial, "n1")
RTH.trivial <- simMatrix(matrix(NA, 6, 2), "n1")
SEM.Mis.Model <- simMisspecSEM(LX = LX.trivial, RTE = RTE.trivial, RTD = RTD.trivial, RTH = RTH.trivial, ex
```

---

simModel *Create a model object*

Description

This function will take model specification from SimSet that contains free parameters, starting values, and fixed values. It will transform the code to a specified SEM package and ready to analyze data.

Usage

```
simModel(object, ...)
```

Arguments

object	SimSet that provides model specification
...	Other values that will be explained specifically for each class

Value

SimModel that will be used for data analysis

Details in ...

- *start:* SimRSet.c that saves all starting values in the model.
- *equalCon:* SimEqualCon.c that save constraints specified by users. The default is no constraint.
- *package:* Desired analysis package
- *estimator:* The default is ML estimator. Other alternatives are GLS, WLS, MLM, MLF, and MLR. Check the sem function help file in the lavaan package for further details
- *auxiliary:* The names or the index of the auxiliary variables in the data
- *indLab:* The names of the variable in the model. The exogenous indicators should be listed first (from x1) and then endogenous indicators should be listed next (from y1).
- *factorLab:* The names of the factors in the model. The exogenous factors should be listed first (from k1) and then endogenous factors should be listed next (from y1).

Author(s)

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

See Also

- SimModel for the simResult
- SimSet for the target object containing model specification

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
LX <- simMatrix(loading, loadingValues)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, 0.5)
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTD <- symMatrix(error.cor)
CFA.Model <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD)
SimModel <- simModel(CFA.Model)

library(lavaan)
loading <- matrix(0, 9, 3)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loading[7:9, 3] <- NA
HS.Model <- simParamCFA(LX = loading)
SimModel <- simModel(HS.Model, indLab=paste("x", 1:9, sep=""))
out <- run(SimModel, HolzingerSwineford1939)
summary(out)
```

SimModel-class *Class* "SimModel"

Description

This class will save information for analysis model and be ready for data analysis.

Objects from the Class

Objects can be created by simModel. It can also be called by new("SimModel", ...).

Slots

modelType: Model type (CFA, Path, or SEM)

param: Set of all free parameters and values of fixed parameters in the model.

start: All starting values of free parameters

`equalCon`: Equality constraints in `SimEqualCon` class

`package`: Packages used in data analysis, either `lavaan` or `OpenMx`. The default is `lavaan`

`estimator`: The default is `ML` estimator. Other alternatives are `GLS`, `WLS`, `MLM`, `MLF`, and `MLR`. Check the sem function help file in the `lavaan` package for further details

`auxiliary`: The names or the index of the auxiliary variables in the data

`indLab`: The names of the variable in the model. The exogenous indicators should be listed first (from x1) and then endogenous indicators should be listed next (from y1).

`factorLab`: The names of the factors in the model. The exogenous factors should be listed first (from k1) and then endogenous factors should be listed next (from y1).

Methods

run To analyze data. There are two required arguments: `object` and `data`. `object` is the `SimModel` object. `data` is data saved in `data.frame`. The following arguments are not required. `simMissing`, is the `SimMissing` object. `indLab` is the labels of the `data` used to name the indicators in the model. If x-side is specified, the x side goes first then the y side. `factorLab` is the labels of the factors in the model. If x-side is specified, the x side goes first and then the y side.

summary To summarize the object

Author(s)

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

See Also

- `simModel` for the constructor of this class.
- `SimEqualCon` for specifying equality constraints.

Examples

```
showClass("SimModel")
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 <- simMatrix(loading, loadingValues)
latent.cor <- matrix(NA, 2, 2)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, 0.5)
error.cor <- matrix(0, 6, 6)
diag(error.cor) <- 1
RTD <- symMatrix(error.cor)
CFA.Model <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD)
SimModel <- simModel(CFA.Model)
summary(SimModel)
```

SimModelMIOut-class *Class* "SimModelMIOut"

Description

This class will save the analysis results from a single analysis using multiple imputation.

Objects from the Class

Objects can be created by run on the SimModel using multiple imputation. It can also be called from the form new("SimModelMIOut", ...).

Slots

param: Set of all free parameters and values of fixed parameters in the model.

start: All starting values of free parameters

equalCon: Equality constraints in SimEqualCon class

package: Packages used in data analysis, either lavaan or OpenMx. The default is lavaan

coef: Parameter estimates saved in matrix arrangement

se: Standard errors of parameter saved in matrix arrangement

fit: Fit Indices values from each replication

converged: Number of convergence replications

paramValue: The parameter values behind the analyzed data.

FMI1: The fraction missing method 1.

FMI2: The fraction missing method 2.

n: Sample size of the analyzed data.

Methods

- summary To summarize the object
- summaryParam To summarize only parameter estimates, standard errors, and significance
- 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 SimModelMIOut objects. See anova for further details.

Author(s)

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

See Also

- SimModel for analysis model
- SimModelOut for the original output model

Examples

```
showClass("SimModelMIOut")
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- simMatrix(loading, 0.7)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 500)
SimModel <- simModel(CFA.Model)
SimMissing <- simMissing(pmMCAR=0.05, numImps=5)
Data <- run(SimData)
Data <- run(SimMissing, Data)
Result <- run(SimModel, Data, SimMissing)
summary(Result)
```

SimModelOut-class *Class* "SimModelOut"

Description

This class will save the analysis results from a single analysis.

Objects from the Class

Objects can be created by run on the SimModel. It can also be called from the form new("SimModelOut", ...).

Slots

param: Set of all free parameters and values of fixed parameters in the model.

start: All starting values of free parameters

equalCon: Equality constraints in SimEqualCon class

package: Packages used in data analysis, either lavaan or OpenMx. The default is lavaan

coef: Parameter estimates saved in matrix arrangement

se: Standard errors of parameter saved in matrix arrangement

fit: Fit Indices values from each replication

converged: Number of convergence replications

paramValue: The parameter values behind the analyzed data.

n: Sample size of the analyzed data.

pMiss: The proportion of missing in each variable.

indLab: Indicator labels of the analyzed data.

factorLab: Factor labels of the analyzed data.

Methods

- summary To summarize the object
- summaryParam To summarize only parameter estimates, standard errors, and significance
- createImpliedMACS To create the model implied means and covariance matrix from the 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 SimModelOut objects. See anova for further details.
- summaryPopulation to summarize the data generation population underlying the analysis.
- getPopulation to extract the data generation population underlying the data used in the analysis.
- setPopulation to put the appropriate data generation model into the analysis result. If the appropriate data generation model is put (the same model as the analysis model), the covarage of population by a confidence interval will be able to be calculated by the summary function. The first argument is the result object. The second argument can be either a matrix set of parameters 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

- SimModel for analysis model

Examples

```
showClass("SimModelOut")
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- simMatrix(loading, 0.7)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 500)
SimModel <- simModel(CFA.Model)
Data <- run(SimData)
Result <- run(SimModel, Data)
summary(Result)
summaryParam(Result)
summaryPopulation(Result)
param <- getPopulation(Result)
Result2 <- setPopulation(Result, param)
Result3 <- setPopulation(Result, CFA.Model)
```

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)
```

SimParam-class *Class* "SimParam"

Description

Set of vectors and matrices that saves free parameters (CFA, Path analysis, or SEM)

Objects from the Class

Object can be created by simParamCFA, simParamPath, or simParamSEM, for CFA, Path analysis, or SEM model, respectively. Objects can be also created by calls of the form new("SimParam", ...).

Slots

modelType: Model type (CFA, Path, or SEM)

LY: Factor loading matrix between endogenous factors and Y indicators

TE: Covariance matrix between Y measurement error

PS: Residual covariance of endogenous factors

BE: Regression effect among endogenous factors

TY: Measurement intercepts of Y indicators

AL: Factor intercepts of endogenous factors

LX: Factor loading matrix between exogenous factors and X indicators

TD: Covariance matrix between X measurement error

PH: Covariance among exogenous factors

GA: Regreeion effect from exogenous factors to endogenous factors

TX: Measurement intercepts of X indicators

KA: Factor Mean of exogenous factors

TH: Measurement error covariance between X indicators and Y indicators

Methods

summary Get the summary of model specification.

extract Extract elements from a simSet. There are several additional arguments. First, if `yOnly` is `TRUE`, then the result will provide only Y side. Second, `y` is the index of indicators in Y side to be extracted. Third, `e` is the index of factors in Y side to be extracted. Fourth, `x` is the index of the indicators in X side to be extracted. Finally, `k` is the index of the factors in X side to be extracted.

Author(s)

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

See Also

- Create an object this class by CFA, Path Analysis, or SEM model by `simParamCFA`, `simParamPath`, or `simParamSEM`, respectively.
- See how to use this object for data analysis by `simModel`.

Examples

```
showClass("SimParam")

library(lavaan)
loading <- matrix(0, 9, 3)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loading[7:9, 3] <- NA
HS.Model <- simParamCFA(LX = loading)
summary(HS.Model)
SimModel <- simModel(HS.Model, indLab=paste("x", 1:9, sep=""))
out <- run(SimModel, HolzingerSwineford1939)
summary(out)

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

---

simParamCFA *Create a set of matrices of parameters for analyzing data that belongs to CFA model.*

Description

This function will create set of matrices of free parameters that belongs to confirmatory factor analysis. The requirement is to specify factor loading matrix.

Usage

```
simParamCFA(...)
```

Arguments

`...`	Each element of model specification, as described in `Details`

Details

NOTE: CFA object can be either specified in X or Y side.

- `LX` or `LY` for factor loading matrix (need to be a `matrix`).
- `TD` or `TE` for measurement error covariance matrix (need to be a `matrix`).
- `PH` or `PS` for factor covariance matrix (need to be a symmetric `matrix`).
- `TX` or `TY` for measurement intercepts (need to be a `vector`).
- `KA`, `AL`, `MK`, or `ME` for factor means (need to be a `vector`).

There are only one required matrices: LY (or LX). The default specifications are

1. The scale-identification default of this model is fixed factor method (factor variances = 1 and factor means = 0).
2. If error covariance matrix is not specified, the default is to estimate all error variances and not estimate error covariances.
3. If factor covariance matrix is not specified, the default is to fix all factor covariance.
4. If factor mean vector is not specified, the default is to fix all factor means to 0.
5. If measurement intercept vector is not specified, the default is to estimate all measurement intercepts.

Value

`SimParam` object that represents the CFA free parameters. This will be used for building `SimModel` later.

Author(s)

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

See Also

- See class `SimParam` for the free parameters object details.
- Use `simParamPath` to specify path analysis model and use `simParamSEM` to specify full structural equation modeling.

Examples

```
loading <- matrix(0, 6, 2)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
CFA.Model <- simParamCFA(LX = loading)
```

simParamPath *Create a set of matrices of parameters for analyzing data that belongs to Path analysis model*

Description

This function will create set of matrices of free parameters that belongs to path analysis model. The requirement is to specify regression coefficient matrix.

Usage

```
simParamPath(..., exo = FALSE)
```

Arguments

...	Each element of model specification, as described in Details
exo	specify TRUE if users wish to specify both exogenous and endogenous indicators.

Details

The matrices and vectors in the endogenous side are

- BE for regression coefficient matrix (need to be a matrix).
- PS for residual covariance matrix (need to be a symmetric matrix).
- AL for indicator intercept (need to be a vector).

The only required matrix for the specification in the endogenous side is BE. If users wish to include the exogenous side in their models, these options are available,

- GA for regression coefficient matrix from exogenous variable to endogenous variable (need to be a matrix).
- PH for exogenous indicator covariance (need to be a symmetric matrix).
- KA or MK for exogenous variable mean (need to be a vector).

The only required matrix for the specification with exogenous side is GA. The default specifications if exo=FALSE are

1. If residual covariance is not specified, then (a) all indicator variances are free, (b) all exogenous covariances are free, (c) all endogenous covariances are fixed.
2. If indicator means vector is not specified, then the indicator means are free.

The default specifications if exo=TRUE are

1. If endogenous indicator covariance (PS) is not specified, then (a) all indicator variances are free, (b) all endogenous indicators covariances are fixed.
2. If endogenous indicators regression coefficient (BE) is not specified, then all coefficients are specified as zero.
3. If indicator means (KA or AL) are not specified, all indictor means are free.
4. If exogenous indicators covariance matrix (PH) is not specified, then the matrix is free in every element.

Value

SimParam object that represents the path analysis model free parameters. This will be used for building SimModel later.

Author(s)

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

See Also

- See class SimParam for the free parameters object details.
- Use simParamCFA to specify CFA model and use simParamSEM to specify full structural equation modeling.

Examples

```
path <- matrix(0, 4, 4)
path[3, 1:2] <- NA
path[4, 3] <- NA
model <- simParamPath(BE=path)

exoPath <- matrix(NA, 3, 2)
model2 <- simParamPath(GA=exoPath, exo=TRUE)
```

---

simParamSEM *Create a set of matrices of parameters for analyzing data that belongs to SEM model*

Description

This function will create set of matrices of free parameters that belongs to full SEM model. The requirement is to specify regression coefficient matrix and factor loading matrix.

Usage

```
simParamSEM(..., exo = FALSE)
```

Arguments

...	Each element of model specification, as described in Details
exo	specify TRUE if users wish to specify both exogenous and endogenous indicators.

Details

The matrices and vectors in the endogenous side are

- LY for factor loading matrix from endogenous factors to Y indicators (need to be a matrix).
- TE for measurement error covariance matrix among Y indicators (need to be a symmetric matrix).
- BE for regression coefficient matrix among endogenous factors (need to be a matrix).

- PS for residual covariance matrix among endogenous factors (need to be a symmetric matrix).
- TY for measurement intercepts of Y indicators. (need to be a vector).
- AL for endogenous factor intercept (need to be a vector).

There are two required matrices for the specification in the endogenous side only: LY and BE. If users need to specify exogenous variable too ("exo=TRUE"), these matrices and vectors are available:

- LX for factor loading matrix from exogenous factors to X indicators (need to be a matrix).
- TD for measurement error covariance matrix among X indicators (need to be a symmetric matrix).
- GA for regression coefficient matrix among exogenous factors (need to be a matrix).
- PH for residual covariance matrix among exogenous factors (need to be a symmetric matrix).
- TX for measurement intercepts of Y indicators. (need to be a vector).
- KA or MK for total mean of exogenous factors (need to be a vector).
- TH for measurement error covariance between X measurement error and Y measurement error (need to be a matrix).

There are four required matrices for the specification in both exogenous and endogenous sides: LY, , BE, LX, and GA. The default specifications if exo=FALSE are

1. If residual factor covariance is not specified, then (a) all factor variances are free, (b) all exogenous covariances are free, (c) all endogenous covariances are fixed.
2. If factor means vector is not specified, then the factor means are free.
3. If error covariance matrix is not specified, the default is to estimate all error variances and not estimate error covariances.
4. If measurement intercept vector is not specified, the default is to estimate all measurement intercepts.

The default specifications if exo=TRUE are

1. If endogenous factor covariance (PS) is not specified, then (a) all endogenous factor variances are free, (b) all endogenous factor covariances are fixed.
2. If endogenous factors regression coefficient (BE) is not specified, then all coefficients are specified as zero.
3. If factor means (KA or AL) are not specified, all indictor means are free.
4. If exogenous factor covariance matrix (PH) is not specified, then the matrix is free in every element.
5. If error covariance matrix (TE, TD, or TH) is not specified, the default is to estimate all error variances and not estimate error covariances.
6. If measurement intercept vector (TX or TY) is not specified, the default is to estimate all measurement intercepts.

Value

SimParam object that represents the path analysis model free parameters. This will be used for building SimModel later.

Author(s)

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

See Also

- See class SimParam for the free parameters object details.
- Use simParamCFA to specify CFA model and use simParamPath to specify path analysis model.

Examples

```
loading <- matrix(0, 8, 3)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loading[7:8, 3] <- NA
path <- matrix(0, 3, 3)
path[3, 1:2] <- NA
SEM.model <- simParamSEM(BE=path, LY=loading)

loading.X <- matrix(0, 6, 2)
loading.X[1:3, 1] <- NA
loading.X[4:6, 2] <- NA
loading.Y <- matrix(NA, 2, 1)
path.GA <- matrix(NA, 1, 2)
BE <- as.matrix(0)
SEM.Exo.model <- simParamSEM(GA=path.GA, BE=BE, LX=loading.X, LY=loading.Y, exo=TRUE)
```

---

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)
```

SimREqualCon-class *Class* "SimREqualCon"

Description

Set of specified equality constraints using the matrix and vector of parameters notation

Objects from the Class

Set of specified equality constraints using the matrix and vector of parameters notation (e.g., no VTE or ME)

Slots

con: List of equality constraint. Each element in the list is an individual equality constraint saved in a matrix. Each row represents each element. If the matrix has two columns, the first column indicates row of the element and the second column indicates column of the element. If the matrix has three columns, the first column is the group of matrix. The rest is row and column. Row name represents the matrix that the element is in. The definition of row name can be seen in VirtualRSet definition.

modelType: Analysis model (CFA, SEM, Path)

Methods

summary Summarize all attributes of this object

Author(s)

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

See Also

- SimEqualCon

Examples

```
# No example
```

simResult *Create simResult.*

Description

This function will create simResult by different ways. One way is to create data and analyze data multiple times by specifying SimData and SimModel and save it in the SimResult.

Usage

```
simResult(nRep = NULL, objData = NULL, objModel = NULL,
objMissing = new("NullSimMissing"), seed = 123321, silent = FALSE,
multicore = FALSE, cluster = FALSE, numProc = NULL, n = NULL,
pmMCAR = NULL, pmMAR = NULL, objSet = NULL,
objFunction = new("NullSimFunction"))
```

Arguments

nRep	Number of replications. Users can specify as NULL and specify n, pmMCAR, and pmMAR as a vector instead. By this, the number of replications will be calculated from the length of n, pmMCAR, and pmMAR.
objData	Data object used in data simulation.
objModel	Model object used in analyzing the simulated data.
objMissing	Model object used in providing the information about missing values.
seed	Seed number. This package will use this seed number to generate the L'Ecuyer (1999) method to make nonoverlapping seed values across replications from the nextRNGStream function.
silent	TRUE if users do not wish to print number of replications during running the function.
multicore	Use multiple processors within a computer. Specify as TRUE to use it.
cluster	Not applicable now. Use for specify nodes in hpc in order to be parallelizable.
numProc	Number of processors for using multiple processors. If it is NULL, the package will find the maximum number of processors.
n	Sample size. This argument is not necessary except the user wish to vary sample size across replications. The sample size here can be random distribution object (VirtualDist), or a vector of sample size in integers. For the random distribution object, if the resulting value has decimal, the value will be rounded.
pmMCAR	The percent completely missing at random. This argument is not necessary except the user wish to vary percent missing completely at random across replications. The pmMCAR here can be random distribution object (VirtualDist), or a vector of percent missing, which the values can be in between 0 and 1 only. The specification of objMissing is not needed (but is needed if users wish to specify complex missing value data generation or wish to use multiple imputation).
pmMAR	The percent missing at random. This argument is not necessary except the user wish to vary percent missing at random across replications. The pmMAR here can be random distribution object (VirtualDist), or a vector of percent missing, which the values can be in between 0 and 1 only. The specification of objMissing is not needed (but is needed if users wish to specify complex missing value data generation or wish to use multiple imputation).

objSet	The SimSet object for data generation and analysis model. Users can specify this argument directly and not specify objData and objModel
objFunction	The function object that will be used for data transformation inside the simulation study. See the example from simFunction

Value

SimResult that saves analysis result from simulate data.

Author(s)

Sunthud Pornprasertmanit (University of Kansas; <psunthud@ku.edu>); Alex Schoemann (University of Kansas; <schoemann@ku.edu>); Patrick Miller (University of Kansas; <patr1ckm@ku.edu>); The L'Ecuyer seed number generation is adapted from the code provided by Paul Johnson in http://winstat.quant.ku.edu/svn/hpcexample/trunk/Ex66-ParallelSeedPrototype/.

References

L'Ecuyer, P. (1999) Good parameters and implementations for combined multiple recursive random number generators. *Operations Research, 47*, 159-164.

See Also

- SimData for data model specification
- SimModel for analysis model specification
- SimResult for the type of resulting object

Examples

```
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- simMatrix(loading, 0.7)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 500)
SimModel <- simModel(CFA.Model)
# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output <- simResult(5, SimData, SimModel)
summary(Output)

# Specify Sample Size by n
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- simMatrix(loading, 0.7)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 500)
SimModel <- simModel(CFA.Model)
# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output <- simResult(NULL, SimData, SimModel, n=seq(50, 100, 10))
summary(Output)
```

```
# Specify both sample size and percent missing completely at random
Output <- simResult(NULL, SimData, SimModel, n=seq(50, 100, 10), pmMCAR=c(0, 0.1, 0.2))
summary(Output)

# Use distribution object on sample size and percent completely at random
n <- simUnif(100, 500)
pmMCAR <- simUnif(0, 0.1)
Output <- simResult(5, SimData, SimModel, n=n, pmMCAR=pmMCAR)
```

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 simResult. 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.

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

- SimData for data generation model.
- SimModel for analysis model
- simResult for the constructor of this class

Examples

```
showClass("SimResult")
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- simMatrix(loading, 0.7)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 500)
SimModel <- simModel(CFA.Model)
# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output <- simResult(5, SimData, SimModel)
summary(Output)
getCutoff(Output, 0.05)
summaryParam(Output)
summaryPopulation(Output)
param <- getPopulation(Output)
Output <- setPopulation(Output, param)
Output2 <- setPopulation(Output, CFA.Model)
```

simResultParam *The constructor of the parameter result object*

Description

This function is used to draw actual and misspecified paraemters in multiple replications. This function will be used to investigate the population misfits.

Usage

```
simResultParam(nRep, object, misspec=new("NullSimMisspec"), SimEqualCon = new("NullSimEqualCon"),
```

Arguments

nRep	The number of drawn (replications)
object	The SimSet class that saves the specification of actual parameters
misspec	The SimMisspec class that saves the specification of misspecified parameters
SimEqualCon	The SimEqualCon object that saves the equalit constraint
seed	The seed number used to draw parameters
maxDraw	The maximum number of sample drawn. This function will draw sets of actual and misspecified parameters repeatedly until the identified sets of actual and misspecified parameters are drawn. The maximum of repetition is specified by this argument.

Value

The parameter result object, SimResultParam

Author(s)

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

Examples

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

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

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

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

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

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

# The number of replications in actual analysis should be much more than 5
ParamObject <- simResultParam(5, Path.Model, Path.Mis.Model)

# Specify the range of misfits to select the set of misspecified parameters
Path.Mis.Model2 <- simMisspecPath(BE = mis.BE, misfitType="rmsea", misfitBound=c(0.05, 0.08))
ParamObject2 <- simResultParam(5, Path.Model, Path.Mis.Model2)

# Find the maximum misspecification for each actual parameter
Path.Mis.Model3 <- simMisspecPath(BE = mis.BE, misfitType="rmsea", optMisfit="max", numIter=10)
ParamObject3 <- simResultParam(5, Path.Model, Path.Mis.Model3)
```

SimResultParam-class *Class* "SimResultParam"

Description

The parameter result object that represents the parameter values used in each replication in a simulation study. This object saves the parameter values and model misspecification of each replication and is able to summarize the population misfits.

Objects from the Class

Object can be created by `simResultParam`, Objects can be also created by calls of the form `new("SimResultParam", ..`

Slots

`modelType`: Model type (CFA, Path, Path.exo, SEM, or SEM.exo)

`nRep`: The number of drawn (replications)

`param`: The `SimSet` class that saves the specification of actual parameters

`misspec`: The `SimMisspec` class that saves the specification of misspecified parameters

`fit`: The population misfits

`seed`: The seed number used to draw parameters

Methods

summary Get the summary of model specification

summaryParam Get the summary of the obtained actual parameters

plotMisfit Plot the population misfit or the relationship between the amount of misfit and the misspecified parameters. See `plotMisfit` for further details.

Author(s)

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

See Also

- simResultParam for the constructor this class
- SimSet for the specification of model parameter
- SimMisspec for the specification of misspecified parameter
- SimEqualCon for the specification of the equality constraints

Examples

```
showClass("SimResultParam")

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

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

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

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

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

mis.path.BE <- matrix(0, 4, 4)
mis.path.BE[4, 1:2] <- NA
mis.BE <- simMatrix(mis.path.BE, "u1")
Path.Mis.Model <- simMisspecPath(BE = mis.BE, misfitType="rmsea") #, misfitBound=c(0.05, 0.08))

# The number of replications in actual analysis should be much more than 5
ParamObject <- simResultParam(5, Path.Model, Path.Mis.Model)
summary(ParamObject)
```

SimSet-class *Class* "SimSet"

Description

Set of vectors and matrices that saves free parameters and parameter values (CFA, Path analysis, or SEM)

Objects from the Class

Object can be created by simSetCFA, simSetPath, or simSetSEM, for CFA, Path analysis, or SEM model, respectively. Objects can be also created by calls of the form new("SimSet", ...).

Slots

`modelType`: Model type (CFA, Path, or SEM)

`LY`: Factor loading matrix between endogenous factors and Y indicators

`TE`: Covariance matrix between Y measurement error

`RTE`: Correlation matrix between Y measurement error

`VTE`: Variance of Y measurement error

`PS`: Residual covariance of endogenous factors

`RPS`: Residual correlation of endogenous factors

`VPS`: Residual variances of endogenous factors

`BE`: Regression effect among endogenous factors

`TY`: Measurement intercepts of Y indicators

`AL`: Factor intercepts of endogenous factors

`ME`: Factor means of endogenous factors

`MY`: Total Mean of Y indicators

`VE`: Total variance of endogenous factors

`VY`: Total variance of Y indicators

`LX`: Factor loading matrix between exogenous factors and X indicators

`TD`: Covariance matrix between X measurement error

`RTD`: Correlation matrix between X measurement error

`VTD`: Variance of X measurement error

`PH`: Covariance among exogenous factors

`RPH`: Correlation among exogenous factors

`GA`: Regression effect from exogenous factors to endogenous factors

`TX`: Measurement intercepts of X indicators

`KA`: Factor Mean of exogenous factors

`MX`: Total Mean of X indicators

`VPH`: Variance of exogenous factors

`VX`: Total variance of X indicators

`TH`: Measurement error covariance between X indicators and Y indicators

`RTH`: Measurement error correlation between X indicators and Y indicators

Methods

run Create a sample of parameters in this object. In other words, draw a sample from all random parameters which is represented in `VirtualDist`.

summary Get the summary of model specification

extract Extract elements from a simSet. There are several additional arguments. First, if `yOnly` is `TRUE`, then the result will provide only Y side. Second, `y` is the index of indicators in Y side to be extracted. Third, `e` is the index of factors in Y side to be extracted. Fourth, `x` is the index of the indicators in X side to be extracted. Finally, `k` is the index of the factors in X side to be extracted.

Author(s)

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

See Also

- Create an object this class by CFA, Path Analysis, or SEM model by simSetCFA, simSetPath, or simSetSEM, respectively.
- See how to specify model misspecification by SimMisspec.

Examples

```
showClass("SimSet")

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

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

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

CFA.Model <- simSetCFA(LX = LX, RPH = RPH, RTD = RTD)
summary(CFA.Model)
#run(CFA.Model)

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

---

simSetCFA *Create a set of matrices of parameter and parameter values to generate and analyze data that belongs to CFA model.*

Description

This function will create set of matrices of free parameters and parameter values that belongs to confirmatory factor analysis. The requirement is to specify factor loading matrix, factor correlation (or covariance) matrix, and error correlation (or covariance) matrix.

Usage

```
simSetCFA(...)
```

Arguments

...	Each element of model specification, as described in Details

Details

NOTE: CFA object can be either specified in X or Y side.

- LX or LY for factor loading matrix (need to be SimMatrix object).
- TD or TE for measurement error covariance matrix (need to be SymMatrix object).
- RTD or RTE for measurement error correlation matrix (need to be SymMatrix object).
- PH or PS for factor covariance matrix (need to be SymMatrix object).
- RPH or RPS for factor correlation matrix (need to be SymMatrix object).
- VTD or VTE for measurement error variance (need to be SimVector object).
- VX or VY for total indicator variance (need to be SimVector object). NOTE: Either measurement error variance or indicator variance is specified. Both cannot be simultaneously specified.
- VPH, VPS, VK, or VE for factor total variance (need to be SimVector object). NOTE: These four objects will have different meanings in simSetSEM function.
- TX or TY for measurement intercepts (need to be SimVector object).
- MX or MY for overall indicator means (need to be SimVector object). NOTE: Either measurement intercept of indicator mean can be specified. Both cannot be specified simultaneously.
- KA, AL, MK, or ME for factor means (need to be SimVector object).

There are three required matrices: LY (or LX), RTE (RTD, TD, or TE), and RPS (RPH, PH, or PS). If users specify the correlation/variance format (instead of the covariance format), the default specifications are

1. All indicator variances are equal to 1. Measurement error variances are automatically implied from total indicator variances.
2. All measurement error variances are free parameters.
3. All indicator means are equal to 0. Indicator intercepts are automatically implied from indicator means.
4. All indicator intercepts are free parameters.
5. All factor variances are equal to 1.
6. All factor variances are fixed.
7. All factor means are equal to 0.
8. All factor means are fixed.

Value

SimSet object that represents the CFA object. This will be used for specifying data or analysis models later.

Author(s)

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

See Also

- See class SimSet for the set of matrices object details.
- See SimMatrix, SymMatrix, or SimVector for input details.
- Use simSetPath to specify path analysis model and use simSetSEM to specify full structural equation modeling.

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

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

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

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

---

simSetPath *Create a set of matrices of parameter and parameter values to generate and analyze data that belongs to Path analysis model*

Description

This function will create set of matrices of free parameters and parameter values that belongs to path analysis model. The requirement is to specify indicator correlation or covariance matrix and regression coefficient matrix.

Usage

```
simSetPath(..., exo = FALSE)
```

Arguments

...	Each element of model specification, as described in Details
exo	specify TRUE if users wish to specify both exogenous and endogenous indicators.

Details

The matrices and vectors in the endogenous side are

- BE for regression coefficient matrix (need to be SimMatrix object).
- PS for residual covariance matrix (need to be SymMatrix object).
- RPS for residual correlation matrix (need to be SymMatrix object).
- VPS for residual indicator variance (need to be SimVector object).
- VE for total indicator variance (need to be SimVector object). NOTE: Either total indicator variance or residual indicator variance is specified. Both cannot be simultaneously specified.
- AL for indicator intercept (need to be SimVector object).
- ME for indicator total mean (need to be SimVector object). NOTE: Either indicator intercept or indicator total mean is specified. Both cannot be simultaneously specified.

There are two required matrices for the specification in the endogenous side only: BE, and RPS (or PS). If users wish to include the exogenous side in their models, these options are available,

- GA for regression coefficient matrix from exogenous variable to endogenous variable (need to be SimMatrix object).
- PH for exogenous factor covariance (need to be SymMatrix object).
- RPH for exogenous factor correlation (need to be SymMatrix object).
- VPH or VK for exogenous variable variance (need to be SimVector object).
- KA or MK for exogenous variable mean (need to be SimVector object). NOTE: Either total indicator variance or residual indicator variance is specified. Both cannot be simultaneously specified.

There are four required matrices for the specification in both exogenous and endogenous sides: BE, RPS (or PS), GA, and RPH (or PH). If users specify the correlation/variance format (instead of the covariance format), the default specifications are

1. All indicator variances are equal to 1. Residual variances are automatically implied from total indicator variances.
2. All residual variances are free parameters.
3. All indicator means are equal to 0. Intercepts are automatically implied from total indicator mean.
4. All indicator intercepts are free parameters.

Value

SimSet object that represents the path analysis simModel. This will be used for specifying data or analysis models later.

Author(s)

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

See Also

- See class SimSet for simResult details.
- See SimMatrix, SymMatrix, or SimVector for input details.
- Use simSetCFA to specify CFA model and use simSetSEM to specify full structural equation modeling.

Examples

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

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

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

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

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

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

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

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

RPS <- symMatrix(diag(2))

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

`simSetSEM` *Create a set of matrices of parameter and parameter values to generate and analyze data that belongs to SEM model*

Description

This function will create set of matrices of free parameters and parameter values that belongs to full SEM model. The requirement is to specify factor residual correlation or covariance matrix, regression coefficient matrix, factor loading matrix, and measurement error correlation or covariance matrix.

Usage

```
simSetSEM(..., exo = FALSE)
```

Arguments

...	Each element of model specification, as described in Details
exo	specify TRUE if users wish to specify both exogenous and endogenous indicators.

Details

The matrices and vectors in the endogenous side are

- LY for factor loading matrix from endogenous factors to Y indicators (need to be SimMatrix object).
- TE for measurement error covariance matrix among Y indicators (need to be SymMatrix object).
- RTE for measurement error correlation matrix among Y indicators (need to be SymMatrix object).
- BE for regression coefficient matrix among endogenous factors (need to be SimMatrix object).
- PS for residual covariance matrix among endogenous factors (need to be SymMatrix object).
- RPS for residual correlation matrix among endogenous factors (need to be SymMatrix object).
- VTE for measurement error variance of Y indicators (need to be SimVector object).
- VY for total variance of Y indicators (need to be SimVector object). NOTE: Either measurement error variance or indicator variance is specified. Both cannot be simultaneously specified.
- TY for measurement intercepts of Y indicators. (need to be SimVector object).
- MY for overall Y indicator means. (need to be SimVector object). NOTE: Either measurement intercept of indicator mean can be specified. Both cannot be specified simultaneously.
- VPS for residual variance of endogenous factors (need to be SimVector object).
- VE for total endogenous factor variance (need to be SimVector object). NOTE: Either total endogenous factor variance or residual endogenous factor variance is specified. Both cannot be simultaneously specified.
- AL for endogenous factor intercept (need to be SimVector object).
- ME for 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.

There are four required matrices for the specification in the endogenous side only: LY, RTE (or TE), BE, and RPS (or PS). If users need to specify exogenous variable too ("exo=TRUE"), these matrices and vectors are available:

- LX for factor loading matrix from exogenous factors to X indicators (need to be SimMatrix object).
- TD for measurement error covariance matrix among X indicators (need to be SymMatrix object).
- RTD for measurement error correlation matrix among X indicators (need to be SymMatrix object).
- GA for regression coefficient matrix among exogenous factors (need to be SimMatrix object).

- PH for residual covariance matrix among exogenous factors (need to be SymMatrix object).
- RPH for residual correlation matrix among exogenous factors (need to be SymMatrix object).
- VTD for measurement error variance of X indicators (need to be SimVector object).
- VX for total variance of X indicators (need to be SimVector object). NOTE: Either measurement error variance or indicator variance is specified. Both cannot be simultaneously specified.
- TX for measurement intercepts of Y indicators. (need to be SimVector object).
- MX for overall Y indicator means. (need to be SimVector object). NOTE: Either measurement intercept of indicator mean can be specified. Both cannot be specified simultaneously.
- VPH or VK for total exogenous factor variance (need to be SimVector object).
- KA or MK for total mean of exogenous factors (need to be SimVector object).
- TH for measurement error covariance between X measurement error and Y measurement error.
- RTH for measurement error correlation between X measurement error and Y measurement error.

There are eight required matrices for the specification in both exogenous and endogenous sides: LY, RTE (or TE), BE, RPS (or PS), LX, RTD (or TD), GA, and RPH (or PH). If users specify the correlation/variance format (instead of the covariance format), the default specifications are

1. All indicator variances are equal to 1. Measurement error variances are automatically implied from total indicator variances.
2. All measurement error variances are free parameters.
3. All indicator means are equal to 0. Indicator intercepts are automatically implied from indicator means.
4. All indicator intercepts are free parameters.
5. All factor variances are equal to 1.
6. All factor variances are fixed.
7. All factor means are equal to 0.
8. All factor means are fixed.

Value

SimSet object that represents the SEM object. This will be used for specifying data or analysis models later.

Author(s)

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

See Also

- See class SimSet for simResult details.
- See SimMatrix, SymMatrix, or SimVector for input details.
- Use simSetCFA to specify CFA model and use simSetPath to specify path analysis model.

Examples

```
u35 <- simUnif(0.3, 0.5)
u68 <- simUnif(0.6, 0.8)
n65 <- simNorm(0.6, 0.05)
loading <- matrix(0, 8, 3)
loading[1:3, 1] <- NA
loading[4:6, 2] <- NA
loading[7:8, 3] <- NA
loading.start <- matrix("", 8, 3)
loading.start[1:3, 1] <- 0.7
loading.start[4:6, 2] <- 0.7
loading.start[7:8, 3] <- "u68"
LY <- simMatrix(loading, loading.start)

RTE <- symMatrix(diag(8))

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)

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

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

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

RTD <- symMatrix(diag(6))

RTE <- symMatrix(diag(2))

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

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

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

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

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

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 *Create simVector that save free parameters and starting values, as well as fixed values*

Description

Create SimVector object that save free parameters and starting values, as well as fixed values. This will be used for model specification later, such as for factor mean vector or measurement error variance vector.

Usage

```
simVector(free = NULL, value = NULL)
```

Arguments

free	Vector of free parameters. Use NA to specify free parameters. Use number as fixed value (including zero). If this argument is not specified, the information from the value argument is used. The positions in the value argument that are 0 or "" are fixed parameters as 0. The other positions are free parameters.
value	Starting values. Can be either one element or vector with the same length as free parameter vector. Each element can be numbers (in either as.numeric or as.character format) or the name of distribution object VirtualDist.

Value

SimVector object that will be used for model specification later.

Author(s)

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

See Also

- See SimVector for the resulting object.
- See simMatrix for creating simMatrix.
- See symMatrix for creating symmetric simMatrix.

Examples

```
factor.mean <- rep(NA, 4)
AL <- simVector(factor.mean, 0)

n02 <- simNorm(0, 0.2)
factor.start <- rep("n02", 4)
KA <- simVector(factor.mean, factor.start)

start <- c(2, 0, 0, 1)
VE <- simVector(value=start)
```

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 simVector function. Objects can be created by calls of the form new("SimVector", ...).

Slots

free: Object of class "vector" draws starting values from the "labels" slot and show as a vector sample.

value: Object of class "vector" provides a thorough description of all information in the object

Methods

adjust Adjust an element in the "SimVector" object

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 and SymMatrix for random parameter symmetric matrix.

Examples

```
showClass("SimVector")

factor.mean <- rep(NA, 2)
factor.mean.starting <- c(5, 2)
AL <- simVector(factor.mean, factor.mean.starting)
run(AL)
summary(AL)
summaryShort(AL)

n01 <- simNorm(0, 1)
AL <- adjust(AL, "n01", 2)
run(AL)
summary(AL)

AL <- extract(AL, 1)
summary(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
```

`standardize` *Standardize the parameter estimates within an object*

Description

Standardize the parameter estimates within an object

Usage

```
standardize(object)
```

Arguments

`object`	The object to be standardized

Value

The object in the same class with standarized values

Methods

signature(object="SimModelOut") This function will extract the coefficients and standardize it

signature(object="SimRSet") This function will extract the coefficients and standardize it

Author(s)

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

Examples

```
# This function is not public.

# 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 <- simMatrix(loading, loadingValues)
# summary(LX)
# 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)
# SimData <- simData(CFA.Model, 200)
# SimModel <- simModel(CFA.Model)
# standardize(run(SimModel, run(SimData)))

# 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 <- simMatrix(loading, loadingValues)
# summary(LX)
# 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)
# set <- reduceMatrices(run(CFA.Model))
```

`startingValues` *Find starting values by averaging random numbers*

Description

Find starting values of free parameters based on pre-specified starting values. If the pre-specified starting values are numbers, the function will use that values. If they are distribution object, this function will randomly draw from the distribution 10 times and take the average of those values.

Usage

```
startingValues(object, trial, ...)
```

Arguments

`object`	The target object that is used to find starting values
`trial`	The number of random drawn to average out and provide the starting values
`...`	Other arguments, such as reduced for reducing X-Y set of matrices to Y set of matrices only

Value

A vector, a matrix, or a `MatrixSet` which includes the starting values of parameters

Methods

signature(object="SimMatrix") Draw samples from the `SimMatrix`, take the average, and report the starting values as a `matrix`.

signature(object="SimVector") Draw samples from the `SimVector`, take the average, and report the starting values as a `vector`.

signature(object="SimSet") Draw samples from `SimSet`, take the average, and report the starting values as a `MatrixSet`.

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)
```

---

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 or linkS4class{SimResultParam} 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.

When `linkS4class{SimResultParam}` is specified in the `object`, the reported fit indices are

- f0 Discrepancy Function
- RMSEA Root Mean Square Error of Approximation
- SRMR Standardized Root Mean Residual

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 `SimResultParam` for the parameter result object input

Examples

```
loading <- matrix(0, 6, 1)
loading[1:6, 1] <- NA
LX <- simMatrix(loading, 0.7)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 500)
SimModel <- simModel(CFA.Model)
# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
Output <- simResult(5, SimData, SimModel)
summaryFit(Output)
summaryFit(Output, detail=TRUE)
```

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	SimResultParam object being described
...	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

SimResultParam for the object input

Examples

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

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

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

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

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

mis.path.BE <- matrix(0, 4, 4)
```

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

# 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 `SimModelOut`, each column means

- `Estimate`: Parameter Estimates
- `SE`: Standard Error of the Parameter Estimates
- `z`: Wald Statistic
- `p`: *p* value based on the Wald Statistic
- `Param`: Parameter Value underlying the analyzed data
- `Bias`: Bias in Parameter Estimates
- `Coverage`: Whether (1-alpha)% confidence interval covers the parameter estimates

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 <- simMatrix(loading, 0.7)
RPH <- symMatrix(diag(1))
RTD <- symMatrix(diag(6))
CFA.Model <- simSetCFA(LY = LX, RPS = RPH, RTE = RTD)
SimData <- simData(CFA.Model, 500)
SimModel <- simModel(CFA.Model)
# We make the examples running only 5 replications to save time.
# In reality, more replications are needed.
```

```
Output <- simResult(5, SimData, SimModel)
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{SimModelOut}, linkS4class{SimDataOut}, or 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{SimModelOut}, linkS4class{SimDataOut}, or 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

- SimDataOut for data output object
- SimModelOut for model output object
- 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

```
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")
summaryShort(LX)
```

`symMatrix` *Create symmetric simMatrix that save free parameters and starting values, as well as fixed values*

Description

Create `SymMatrix` object that save free parameters and starting values, as well as fixed values. This will be used for model specification later, such as for factor residual correlation matrix or measurement error correlation matrix.

Usage

```
symMatrix(free = NULL, value = NULL)
```

Arguments

`free`	Symmetric matrix of free parameters. Use `NA` to specify free parameters. Use number as fixed value (including zero). The input matrix need to be symmetric matrix. If this argument is not specified, the information from the `value` argument is used. The positions in the `value` argument that are 0 or "" are fixed parameters as 0. The positions with 1 are fixed parameters as 1. The other positions are free parameters.
`value`	Starting values. Can be either one element or matrix with the same dimension as free parameter matrix. Each element can be numbers (in either `as.numeric` or `as.character` format) or the name of distribution object `VirtualDist`.

Value

`SymMatrix` object that will be used for model specification later.

Author(s)

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

See Also

See `VirtualDist` for the resulting object. See `simMatrix` for creating simMatrix and `simVector` for simVector.

Examples

```
latent.cor <- matrix(NA, 3, 3)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, 0.5)

u46 <- simUnif(0.4, 0.6)
factor.cor <- matrix(NA, 4, 4)
diag(factor.cor) <- 1
factor.cor.start <- matrix("u46", 4, 4)
factor.cor.start[1, 2] <- factor.cor.start[2, 1] <- "0.5"
RPS <- symMatrix(factor.cor, factor.cor.start)
```

```
start <- diag(4)
start[1, 2] <- 0.5
start[2, 1] <- 0.5
ST <- symMatrix(value=start)
```

SymMatrix-class *Symmetric matrix object: Random parameters symmetric matrix*

Description

This object can be used to represent a symmetric matrix in SEM model. It contains free parameters, fixed values, and starting values. This object can be represented factor correlation or error correlation matrix.

Objects from the Class

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

Slots

free: indicates which elements of the matrix are free or fixed. "NA" means the element is freely estimated. Numbers (including 0) means the element is fixed to be the indicated number.

value: indicates the starting values of each element in the matrix. The starting values could be numbers or the name of "distribution objects"

Extends

Class "SimMatrix", directly.

Methods

adjust Adjust an element in the "SymMatrix" object

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

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 and SimVector for random parameter vector.

Examples

```
showClass("SymMatrix")

latent.cor <- matrix(NA, 3, 3)
diag(latent.cor) <- 1
RPH <- symMatrix(latent.cor, 0.5)

u46 <- simUnif(0.4, 0.6)
RPH <- adjust(RPH, "u46", c(3,2))
summary(RPH)
summaryShort(RPH)
run(RPH)
```

---

tagHeaders *Tag names to each element*

Description

This element of a vector will be tagged by the names of the vector with the position of the element. This element of a matrix will be tagged by the names of the matrix with the row and column positions of the matrix.

Usage

```
tagHeaders(object, ...)
```

Arguments

object	The object to be tagged
...	The additional arguments

Value

The object with the row, column, or element names.

Methods

signature(object="VirtualRSet") This element of a vector will be tagged by the names of the vector with the position of the element. This element of a matrix will be tagged by the names of the matrix with the row and column positions of the matrix. *Y* means indicators on *Y*-side. *X* means indicators on *X*-side. *E* means endogenous factors. *K* means exogenous factors.

Author(s)

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

Examples

```
# No example
```

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)
```

toSimSet *Transform the analysis model object into the object for data generation*

Description

Transform an analysis model object (SimModelOut) or reduced parameters object (SimRSet) into the object for data generation (SimSet)

Usage

```
toSimSet(out, ...)
```

Arguments

out	The analysis output object to be transformed (SimModelOut)
...	An additional argument

Value

The SimSet that contains parameters for data generation.

Methods

signature(out = "SimModelOut") This function transforms an analysis model object into the object for data generation. The additional argument is usedStd. If usedStd=TRUE, the standardized estimates will be used. If usedStd=FALSE, the unstandardized estimates will be used.

signature(out = "SimRSet") This function transforms a parameter value object (in data analysis parameterization) into the object for data generation. The additional argument is param, which is the SimParam saving the free parameters specification.

Author(s)

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

Examples

```
# This function is not public.

# library(lavaan)
# hs <- HolzingerSwineford1939
# 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=""))
# out <- run(SimModel, hs)
# set <- toSimSet(out)
```

---

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(object)
```

Arguments

`object`	A target `MatrixSet`

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
```

---

vectorizeObject *Change an object to a vector with labels*

Description

Change an object to a vector with labels

Usage

```
vectorizeObject(object, labels, ...)
```

Arguments

object	The object to be vectorized
labels	The labels of each element in the object
...	The other additional arguments, such as whether an object is symmetric matrix

Value

A vector with labels

Methods

signature(object="vector", labels="vector") This function will select elements in the object that the corresponding elements in labels are not NA and give the name for it.

signature(object="matrix", labels="matrix") This function will select elements in the object that the corresponding elements in labels are not NA, give the name for it, and then transform it to vector. The additional arguement is symmetric, which is TRUE if the matrix is symmetric matrix

signature(object="VirtualRSet", labels="SimLabels") This function will vectorize every matrix or vector in the object and combine them together to a single vector.

signature(object="MatrixSet", labels="SimGenLabels") This function will vectorize every matrix or vector in the object and combine them together to a single vector.

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 simMatrix, symMatrix, simVector, and simResult (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 put in setting the data object up, simData.

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

- simMatrix Random parameter matrix. A distribution object can be used to create random parameter.
- symMatrix Random parameter symmetric matrix. A distribution object can be used to create random parameter.
- simVector Random parameter vector. A distribution object can be used to create random parameter.
- simResult 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))
```

`writeLavaanCode` *Write a lavaan code given the matrices of free parameter*

Description

Write a lavaan code given the matrices of free parameter

Usage

```
writeLavaanCode(object, constraint, aux = NULL)
```

Arguments

`object`	A parameter object, `SimParam`
`constraint`	An equality constraint
`aux`	The names of auxiliary variables

Value

A string of lavaan code

Author(s)

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

Examples

```
# No example
```

writeLavaanConstraint *Write a lavaan code for a given set of equality constraints*

Description

Write a lavaan code for a given set of equality constraints

Usage

```
writeLavaanConstraint(object, constraint)
```

Arguments

object	A parameter object, SimParam
constraint	An equality constraint

Value

A SimRSet class with the text of lavaan constraint

Author(s)

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

Examples

```
# No example
```

writeLavaanIndividualConstraint *Write a lavaan code for a given equality constraint for each parameter*

Description

Write a lavaan code for a given equality constraint for each parameter

Usage

```
writeLavaanIndividualConstraint(Matrix, Attribute, Names)
```

Arguments

Matrix	The name of a matrix or a vector
Attribute	A row in each equality constraint matrix ([group], [row], [column]) or ([group], [element])
Names	A matrix or a vector that contains row and column names for indicator or factor labels

Value

A string for the specification of equality constraint in `lavaan`

Author(s)

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

Examples

```
# No example
```

`writeLavaanNullCode` *Write a lavaan code for a null model*

Description

Write a lavaan code for a null model

Usage

```
writeLavaanNullCode(var, aux = NULL)
```

Arguments

`var`	The name of variables in the model
`aux`	The name of auxiliary variables

Value

A string containing the `lavaan` code for a null model

Author(s)

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

Examples

```
# No example
```

Index

∗Topic **classes**
Nullclass, 82
SimData-class, 139
SimDataDist-class, 141
SimDataOut-class, 143
SimEqualCon-class, 146
SimFunction-class, 150
SimGenLabels-class, 153
SimMatrix-class, 158
SimMissing-class, 161
SimMisspec-class, 162
SimModel-class, 170
SimModelMIOut-class, 172
SimModelOut-class, 173
SimREqualCon-class, 183
SimResult-class, 186
SimResultParam-class, 189
SimVector-class, 202
SymMatrix-class, 216
VirtualDist-class, 222
∗Topic **run**
run, 122

adjust, 5, *158*, *202*, *216*
adjust,ANY-method *(*adjust*)*, 5
adjust,SimMatrix-method *(*adjust*)*, 5
adjust,SimVector-method *(*adjust*)*, 5
adjust,SymMatrix-method *(*adjust*)*, 5
adjust-methods *(*adjust*)*, 5
all.equal, *82*
amelia, *128*, *160*
anova, 7, *172*, *174*, *187*
anova,SimModelMIOut-method *(*anova*)*, 7
anova,SimModelOut-method *(*anova*)*, 7
anova,SimResult-method *(*anova*)*, 7

blankParameters, 8

centralMoment, 9
checkInputValue, 9
checkInputValueVector *(*checkInputValue*)*, 9
clean, 10
cleanSimResult, 11
collapseExo, 11
combineLatentCorExoEndo, 12
combineLoadingExoEndo, 13
combineMeasurementErrorExoEndo, 13
combineObject, 14
combineObject,ANY,ANY-method *(*combineObject*)*, 14
combineObject,matrix,matrix-method *(*combineObject*)*, 14
combineObject,MatrixSet,MatrixSet-method *(*combineObject*)*, 14
combineObject,SimMatrix,SimMatrix-method *(*combineObject*)*, 14
combineObject,SimParam,list-method *(*combineObject*)*, 14
combineObject,SimVector,SimVector-method *(*combineObject*)*, 14
combineObject,vector,vector-method *(*combineObject*)*, 14
combineObject-methods *(*combineObject*)*, 14
combinePathExoEndo, 15
constantVector, 16
constrainMatrices, 16
constrainMatrices,ANY,ANY-method *(*constrainMatrices*)*, 16
constrainMatrices,MatrixSet,SimEqualCon-method *(*constrainMatrices*)*, 16
constrainMatrices-methods *(*constrainMatrices*)*, 16
continuousPower, 17, *42*, *43*, *45*, *110*
countFreeParameters, 19
countFreeParameters,ANY-method *(*countFreeParameters*)*, 19
countFreeParameters,matrix-method *(*countFreeParameters*)*, 19
countFreeParameters,SimEqualCon-method *(*countFreeParameters*)*, 19
countFreeParameters,SimMatrix-method *(*countFreeParameters*)*, 19
countFreeParameters,SimREqualCon-method *(*countFreeParameters*)*, 19
countFreeParameters,SimSet-method

(countFreeParameters), 19
countFreeParameters,SimVector-method (countFreeParameters), 19
countFreeParameters,SymMatrix-method (countFreeParameters), 19
countFreeParameters,vector-method (countFreeParameters), 19
countFreeParameters,VirtualRSet-method (countFreeParameters), 19
countFreeParameters-methods (countFreeParameters), 19
countMACS, 20
cov2corMod, 20
createData, 21
createFreeParameters, 21
createImpliedMACS, 22, *143*, *174*
createImpliedMACS,MatrixSet-method (createImpliedMACS), 22
createImpliedMACS,SimDataOut-method (SimDataOut-class), 143
createImpliedMACS,SimModelOut-method (SimModelOut-class), 173
createImpliedMACS,SimRSet-method (createImpliedMACS), 22
createImpliedMACS-methods (createImpliedMACS), 22

defaultStartingValues, 23
distribution object, *6*
divideObject, 24
divideObject,ANY-method (divideObject), 24
divideObject,matrix,numeric-method (divideObject), 24
divideObject,MatrixSet,numeric-method (divideObject), 24
divideObject,vector,numeric-method (divideObject), 24
divideObject-methods (divideObject), 24
drawParameters, 25
drawParametersMisspec, 25

expandMatrices, 26
extract, 27, *158*
extract,data.frame-method (extract), 27
extract,matrix-method (extract), 27
extract,SimDataDist-method (SimDataDist-class), 141
extract,SimMatrix-method (SimMatrix-class), 158
extract,SimSet-method (SimSet-class), 190
extract,SimVector-method (SimVector-class), 202
extract,vector-method (extract), 27
extract,VirtualRSet-method (extract), 27
extract-methods (extract), 27
extractLavaanFit, 28
extractMatrixNames, 28
extractOpenMxFit, 29
extractVectorNames, 29

fillParam, 30
find2Dhist, 31
findFactorIntercept, 31, *33–40*
findFactorMean, *32*, 32, *34–40*
findFactorResidualVar, *32, 33*, 33, *35–40*
findFactorTotalCov, *32–34*, 34, *36–40*
findFactorTotalVar, *32–35*, 36, *37–40*
findIndIntercept, *32–36*, 37, *38–40*
findIndMean, *32–37*, 38, *39, 40*
findIndResidualVar, *32–38*, 39, *40*
findIndTotalVar, *32–39*, 40
findphist, 41
findPossibleFactorCor, 41, *44*
findPower, 42, *45*
findRecursiveSet, *42*, 43, *44*
findRowZero, 44
findTargetPower, 45
fitMeasuresChi, 46
freeVector, 46

getCondQtile, 47
getCutoff, *48*, 48, *50*, *52*, *57, 58*, *60*, *86*, *97*, *124*, *186*
getCutoff,data.frame-method (getCutoff), 48
getCutoff,matrix-method (getCutoff), 48
getCutoff,SimResult-method (getCutoff), 48
getCutoff-methods (getCutoff), 48
getCutoffNested, 49, *52*, *88*, *100*
getCutoffNonNested, 51, *62*, *89*, *102*
getKeywords, 53
getPopulation, 54, *143*, *174*, *187*
getPopulation,ANY-method (getPopulation), 54
getPopulation,SimDataOut-method (SimDataOut-class), 143
getPopulation,SimModelOut-method (SimModelOut-class), 173
getPopulation,SimResult-method (SimResult-class), 186
getPopulation-methods (getPopulation), 54

getPower, *42*, *43*, *45*, 55, *96*, *110*
getPowerFit, 57, *97*, *186*
getPowerFit,data.frame,vector-method (getPowerFit), 57
getPowerFit,matrix,vector-method (getPowerFit), 57
getPowerFit,SimResult,missing-method (getPowerFit), 57
getPowerFit,SimResult,vector-method (getPowerFit), 57
getPowerFit-methods (getPowerFit), 57
getPowerFitNested, 59, *100*
getPowerFitNested,SimResult,SimResult,missing-method (getPowerFitNested), 59
getPowerFitNested,SimResult,SimResult,vector-method (getPowerFitNested), 59
getPowerFitNested-methods (getPowerFitNested), 59
getPowerFitNonNested, 61, *103*
getPowerFitNonNested,SimResult,SimResult,missing-method (getPowerFitNonNested), 61
getPowerFitNonNested,SimResult,SimResult,vector-method (getPowerFitNonNested), 61
getPowerFitNonNested-methods (getPowerFitNonNested), 61

imposeMissing, 63, *160*, *162*
indProd, 65
interpolate, 66
isCorMatrix, 67
isDefault, 68
isMeanConstraint, 68
isNullObject, 69
isNullObject,ANY,ANY-method (isNullObject), 69
isNullObject,data.frame-method (isNullObject), 69
isNullObject,matrix-method (isNullObject), 69
isNullObject,SimDataDist-method (isNullObject), 69
isNullObject,SimEqualCon-method (isNullObject), 69
isNullObject,SimFunction-method (isNullObject), 69
isNullObject,SimMatrix-method (isNullObject), 69
isNullObject,SimMissing-method (isNullObject), 69
isNullObject,SimMisspec-method (isNullObject), 69
isNullObject,SimREqualCon-method (isNullObject), 69
isNullObject,SimSet-method (isNullObject), 69
isNullObject,SimVector-method (isNullObject), 69
isNullObject,SymMatrix-method (isNullObject), 69
isNullObject,vector-method (isNullObject), 69
isNullObject,VirtualRSet-method (isNullObject), 69
isNullObject-methods (isNullObject), 69
isRandom, 70
isRandom,ANY-method (isRandom), 70
isRandom,SimMatrix-method (isRandom), 70
isRandom,SimSet-method (isRandom), 70
isRandom,SimVector-method (isRandom), 70
isRandom-methods (isRandom), 70
isVarianceConstraint, 71

kiStat, 71
kurtosis, 72, *223*
kurtosis,vector-method (kurtosis), 72
kurtosis,VirtualDist-method (VirtualDist-class), 222
kurtosis-methods (kurtosis), 72

likRatioFit, 73
loadingFromAlpha, 74

makeLabels, 75, *153*
makeLabels,ANY-method (makeLabels), 75
makeLabels,matrix-method (makeLabels), 75
makeLabels,SimParam-method (makeLabels), 75
makeLabels,SimSet-method (makeLabels), 75
makeLabels,vector-method (makeLabels), 75
makeLabels,VirtualDist-method (makeLabels), 75
makeLabels-methods (makeLabels), 75
matchKeywords, 76
MatrixSet, *24*, *26*, *120*, *154*, *206*, *220*
MatrixSet-class, 77
miPool, 78, *80*, *81*, *129*
miPoolChi, *79*, 79
miPoolVector, *79*, *80*, 80
MisspecSet-class (MatrixSet-class), 77
multipleAllEqual, 82

nextRNGStream, *184*
Nullclass, 82

NullDataFrame, *69*
NullDataFrame-class *(*Nullclass*)*, 82
NullMatrix, *24*, *69*
NullMatrix-class *(*Nullclass*)*, 82
NullRSet-class *(*Nullclass*)*, 82
NullSimDataDist, *69*
NullSimDataDist-class *(*Nullclass*)*, 82
NullSimEqualCon, *69*
NullSimEqualCon-class *(*Nullclass*)*, 82
NullSimFunction, *69*
NullSimFunction-class *(*Nullclass*)*, 82
NullSimMatrix, *69*
NullSimMatrix-class *(*Nullclass*)*, 82
NullSimMissing, *69*
NullSimMissing-class *(*Nullclass*)*, 82
NullSimMisspec, *69*
NullSimMisspec-class *(*Nullclass*)*, 82
NullSimREqualCon, *69*
NullSimREqualCon-class *(*Nullclass*)*, 82
NullSimSet, *69*
NullSimSet-class *(*Nullclass*)*, 82
NullSimVector, *69*
NullSimVector-class *(*Nullclass*)*, 82
NullSymMatrix, *69*
NullSymMatrix-class *(*Nullclass*)*, 82
NullVector, *24*, *69*
NullVector-class *(*Nullclass*)*, 82

overlapHist, 83

ParameterSet, 84
plot3DQtile, 85
plotCutoff, 86, *124*, *186*
plotCutoff,data.frame-method *(*plotCutoff*)*, 86
plotCutoff,SimResult-method *(*plotCutoff*)*, 86
plotCutoff-methods *(*plotCutoff*)*, 86
plotCutoffNested, 87, *100*
plotCutoffNonNested, *52*, 89, *102*
plotDist, 90, *223*
plotDist,SimDataDist-method *(*SimDataDist-class*)*, 141
plotDist,VirtualDist-method *(*VirtualDist-class*)*, 222
plotDist-methods *(*plotDist*)*, 90
plotIndividualScatter, 91
plotLogisticFit, 92
plotMisfit, 93, *189*
plotOverHist, 94
plotPower, 95, *104*
plotPowerFit, *92*, *93*, *95*, 96, *98*, *99*, *106*, *187*
plotPowerFitDf, 98
plotPowerFitNested, 99
plotPowerFitNonNested, 101
plotPowerSig, 103
plotQtile, 104
plotScatter, 105
popDiscrepancy, 106, *108*, *109*, *209*
popMisfit, 107
popMisfit,ANY,ANY-method *(*popMisfit*)*, 107
popMisfit,list,list-method *(*popMisfit*)*, 107
popMisfit,matrix,matrix-method *(*popMisfit*)*, 107
popMisfit,MatrixSet,MatrixSet-method *(*popMisfit*)*, 107
popMisfit,SimRSet,SimRSet-method *(*popMisfit*)*, 107
popMisfit,SimSet,SimMisspec-method *(*popMisfit*)*, 107
popMisfit-methods *(*popMisfit*)*, 107
popMisfitMACS, *107*, 108, *163*, *165*, *166*, *168*, *209*
predProb, 109
printIfNotNull, 110
pValue, 111, *113*, *118*, *124*
pValue,ANY-method *(*pValue*)*, 111
pValue,numeric,data.frame-method *(*pValue*)*, 111
pValue,numeric,vector-method *(*pValue*)*, 111
pValue,SimModelOut,SimResult-method *(*pValue*)*, 111
pValue-methods *(*pValue*)*, 111
pValueCondCutoff, 113
pValueNested, *73*, 114
pValueNonNested, *73*, 115
pValueVariedCutoff, 117

reassignNames, 118
reduceConstraint, 119
reduceMatrices, 120
residualCovariate, 120
revText, 121
run, 122, *143*, *150*, *158*, *162*, *172*, *173*, *202*, *216*, *223*
run,ANY-method *(*run*)*, 122
run,NullSimMatrix-method *(*run*)*, 122
run,NullSimVector-method *(*run*)*, 122
run,NullSymMatrix-method *(*run*)*, 122
run,SimBeta-method *(*VirtualDist-class*)*, 222
run,SimBinom-method *(*VirtualDist-class*)*, 222

run,SimCauchy-method (VirtualDist-class), 222
run,SimChisq-method (VirtualDist-class), 222
run,SimData-method (SimData-class), 139
run,SimDataDist-method (SimDataDist-class), 141
run,SimExp-method (VirtualDist-class), 222
run,SimF-method (VirtualDist-class), 222
run,SimFunction-method (SimFunction-class), 150
run,SimGamma-method (VirtualDist-class), 222
run,SimGenLabels-method (SimGenLabels-class), 153
run,SimGeom-method (VirtualDist-class), 222
run,SimHyper-method (VirtualDist-class), 222
run,SimLnorm-method (VirtualDist-class), 222
run,SimLogis-method (VirtualDist-class), 222
run,SimMatrix-method (SimMatrix-class), 158
run,SimMissing-method (SimMissing-class), 161
run,SimMisspec-method (SimMisspec-class), 162
run,SimModel-method (SimModel-class), 170
run,SimNbinom-method (VirtualDist-class), 222
run,SimNorm-method (VirtualDist-class), 222
run,SimPois-method (VirtualDist-class), 222
run,SimSet-method (SimSet-class), 190
run,SimT-method (VirtualDist-class), 222
run,SimUnif-method (VirtualDist-class), 222
run,SimVector-method (SimVector-class), 202
run,SimWeibull-method (VirtualDist-class), 222
run,SymMatrix-method (SymMatrix-class), 216
run-methods (run), 122
runFit, *112*, *115*, *117*, 123, *126*
runFit,ANY-method (runFit), 123
runFit,SimModel-method (runFit), 123
runFit,SimModelOut-method (runFit), 123
runFit-methods (runFit), 123
runFitParam, 126
runFitParam,ANY-method (runFitParam), 126
runFitParam,SimModel-method (runFitParam), 126
runFitParam,SimModelOut-method (runFitParam), 126
runFitParam-methods (runFitParam), 126
runLavaan, 127
runMI, *65*, *79*, *81*, 128, *160*
runMisspec, 129, *144*, *163*, *165*, *166*, *168*
runRep, 131

setOpenMxObject, 132
setOpenMxObject,ANY,ANY-method (setOpenMxObject), 132
setOpenMxObject,matrix,matrix-method (setOpenMxObject), 132
setOpenMxObject,SimParam,SimRSet-method (setOpenMxObject), 132
setOpenMxObject,vector,vector-method (setOpenMxObject), 132
setOpenMxObject-methods (setOpenMxObject), 132
setPopulation, 133, *174*, *187*
setPopulation,ANY-method (setPopulation), 133
setPopulation,SimModelOut,SimRSet-method (SimModelOut-class), 173
setPopulation,SimModelOut,SimSet-method (SimModelOut-class), 173
setPopulation,SimResult,data.frame-method (SimResult-class), 186
setPopulation,SimResult,SimSet-method (SimResult-class), 186
setPopulation,SimResult,VirtualRSet-method (SimResult-class), 186
setPopulation-methods (setPopulation), 133
SimBeta, *134*, *223*
simBeta, 134, *223*
SimBeta-class (VirtualDist-class), 222
SimBinom, *135*, *223*
simBinom, 134, *223*
SimBinom-class (VirtualDist-class), 222
SimCauchy, *135*, *223*
simCauchy, 135, *223*
SimCauchy-class (VirtualDist-class), 222
SimChisq, *136*, *223*
simChisq, 136, *223*
SimChisq-class (VirtualDist-class), 222

SimData, *21*, *25*, *70*, *123*, *131*, *137*, *140, 141*, *143*, *184, 185*, *187*
simData, *124*, 137, *139*, *146*, *222*
simData,ANY-method *(*simData*)*, 137
simData,SimModelOut-method *(*simData*)*, 137
simData,SimRSet-method *(*simData*)*, 137
simData,SimSet-method *(*simData*)*, 137
SimData-class, 139
simData-methods *(*simData*)*, 137
SimDataDist, *27*, *69*, *83*, *91*, *124*, *137*, *139–141*
simDataDist, *138*, 140, *141, 142*, *222*, *224*
SimDataDist-class, 141
SimDataOut, *21*, *55*, *139*, *213*
SimDataOut-class, 143
SimEqualCon, *26*, *69*, *82*, *119*, *139*, *143*, *145*, *171–173*, *183*, *188*, *190*
simEqualCon, *137, 138*, 144, *146*
SimEqualCon-class, 146
SimExp, *147*, *224*
simExp, 147, *223*
SimExp-class *(*VirtualDist-class*)*, 222
SimF, *148*, *224*
simF, 148, *223*
SimF-class *(*VirtualDist-class*)*, 222
SimFunction, *69*, *83*, *131*, *149*
simFunction, *66*, *121*, 148, *150*, *185*
SimFunction-class, 150
SimGamma, *152*, *224*
simGamma, 152, *223*
SimGamma-class *(*VirtualDist-class*)*, 222
SimGenLabels-class, 153
SimGeom, *154*, *224*
simGeom, 154, *223*
SimGeom-class *(*VirtualDist-class*)*, 222
SimHyper, *155*, *224*
simHyper, 155, *223*
SimHyper-class *(*VirtualDist-class*)*, 222
SimLabels, *84*
SimLabels-class *(*ParameterSet*)*, 84
SimLnorm, *156*, *224*
simLnorm, 155, *223*
SimLnorm-class *(*VirtualDist-class*)*, 222
SimLogis, *156*, *224*
simLogis, 156, *223*
SimLogis-class *(*VirtualDist-class*)*, 222
SimMatrix, *5, 6*, *14*, *27*, *69, 70*, *82*, *123*, *157*, *162*, *193–195*, *197, 198*, *202*, *206*, *214*, *216*
simMatrix, 157, *158*, *202*, *215*, *222*, *224*
SimMatrix-class, 158
SimMissing, *65*, *69*, *83*, *131*, *159, 160*
simMissing, 159
SimMissing-class, 161
SimMisspec, *25, 26*, *69, 70*, *78*, *83*, *123*, *129*, *139*, *165*, *167, 168*, *188–190*, *192*
SimMisspec-class, 162
simMisspecCFA, *123*, *126*, *137, 138*, *162*, *164*, 164, *167, 168*
simMisspecPath, *123*, *126*, *137, 138*, *162*, *164, 165*, 166, *168*
simMisspecSEM, *123*, *126*, *137, 138*, *162*, *164, 165*, *167*, 167
SimModel, *122–127*, *131*, *140, 141*, *143*, *169, 170*, *172–174*, *178*, *180, 181*, *184, 185*, *187*
simModel, *146*, 169, *170, 171*, *177*
simModel,ANY-method *(*simModel*)*, 169
simModel,SimModelOut-method *(*simModel*)*, 169
simModel,SimParam-method *(*simModel*)*, 169
simModel,SimSet-method *(*simModel*)*, 169
SimModel-class, 170
simModel-methods *(*simModel*)*, 169
SimModelMIOut-class, 172
SimModelOut, *7*, *55*, *73*, *81*, *111, 112*, *114–117*, *127*, *133*, *138*, *172*, *211*, *213*, *218*
SimModelOut-class, 173
SimNbinom, *175*, *224*
simNbinom, 175, *223*
SimNbinom-class *(*VirtualDist-class*)*, 222
SimNorm, *123*, *176*, *224*
simNorm, 175, *223*
SimNorm-class *(*VirtualDist-class*)*, 222
SimParam, *15*, *21–23*, *84*, *132*, *178*, *180–182*, *219*, *226, 227*
SimParam-class, 176
simParamCFA, *176, 177*, 177, *180*, *182*
simParamPath, *176–178*, 179, *182*
simParamSEM, *176–178*, *180*, 180
SimPois, *182*, *224*
simPois, 182, *223*
SimPois-class *(*VirtualDist-class*)*, 222
SimREqualCon, *69*, *82*, *119*
SimREqualCon-class, 183
SimResult, *7*, *10, 11*, *17, 18*, *48–52*, *55–62*, *73*, *86–89*, *95–97*, *99, 100*, *102*, *111, 112*, *114–117*, *123–126*, *133*, *141*, *149*, *151*, *184, 185*, *208, 209*, *211–213*
simResult, *65*, *131*, *149, 150*, *159, 160*, 184, *186, 187*, *222*, *224*

SimResult-class, 186
SimResultParam, *93*, *126*, *188*, *209, 210*
simResultParam, 188, *189, 190*
SimResultParam-class, 189
SimRSet, *15*, *23*, *26*, *120*, *130*, *132*, *218*, *227*
SimRSet-class *(*ParameterSet*)*, 84
SimSet, *20, 21*, *25–27*, *69, 70*, *78*, *82*, *119*, *123*, *129*, *139*, *143*, *164*, *169, 170*, *185*, *188–190*, *193–195*, *198*, *206*, *218, 219*
SimSet-class, 190
simSetCFA, *137, 138*, *145*, *164, 165*, *190*, *192*, 192, *195*, *198*
simSetPath, *137, 138*, *145*, *166, 167*, *190*, *192*, *194*, 194, *198*
simSetSEM, *137, 138*, *145*, *167, 168*, *190*, *192*, *194, 195*, 196
SimT, *200*, *224*
simT, 200, *223*
SimT-class *(*VirtualDist-class*)*, 222
SimUnif, *123*, *201*, *224*
simUnif, 200, *223*
SimUnif-class *(*VirtualDist-class*)*, 222
SimVector, *5, 6*, *14*, *16*, *27*, *46, 47*, *69, 70*, *82*, *123*, *159*, *162*, *193–195*, *197, 198*, *201, 202*, *206*, *214*, *216*
simVector, *157*, 201, *202*, *215*, *222*, *224*
SimVector-class, 202
SimWeibull, *203*, *224*
simWeibull, 203, *223*
SimWeibull-class *(*VirtualDist-class*)*, 222
skew, 204, *223*
skew,vector-method *(*skew*)*, 204
skew,VirtualDist-method *(*VirtualDist-class*)*, 222
skew-methods *(*skew*)*, 204
sortList, 204
standardize, 205
standardize,ANY-method *(*standardize*)*, 205
standardize,SimModelOut-method *(*standardize*)*, 205
standardize,SimRSet-method *(*standardize*)*, 205
standardize-methods *(*standardize*)*, 205
startingValues, 206
startingValues,ANY-method *(*startingValues*)*, 206
startingValues,SimMatrix-method *(*startingValues*)*, 206
startingValues,SimSet-method *(*startingValues*)*, 206
startingValues,SimVector-method *(*startingValues*)*, 206
startingValues-methods *(*startingValues*)*, 206
subtractObject, 207
subtractObject,ANY,ANY-method *(*subtractObject*)*, 207
subtractObject,SimRSet,SimRSet-method *(*subtractObject*)*, 207
subtractObject-methods *(*subtractObject*)*, 207
summary, *133*, *143*, *150*, *162*, *172*, *174*, *187*, *202*
summary,MatrixSet-method *(*MatrixSet-class*)*, 77
summary,MisspecSet-method *(*MatrixSet-class*)*, 77
summary,SimBeta-method *(*VirtualDist-class*)*, 222
summary,SimBinom-method *(*VirtualDist-class*)*, 222
summary,SimCauchy-method *(*VirtualDist-class*)*, 222
summary,SimChisq-method *(*VirtualDist-class*)*, 222
summary,SimData-method *(*SimData-class*)*, 139
summary,SimDataDist-method *(*SimDataDist-class*)*, 141
summary,SimDataOut-method *(*SimDataOut-class*)*, 143
summary,SimEqualCon-method *(*SimEqualCon-class*)*, 146
summary,SimExp-method *(*VirtualDist-class*)*, 222
summary,SimF-method *(*VirtualDist-class*)*, 222
summary,SimFunction-method *(*SimFunction-class*)*, 150
summary,SimGamma-method *(*VirtualDist-class*)*, 222
summary,SimGeom-method *(*VirtualDist-class*)*, 222
summary,SimHyper-method *(*VirtualDist-class*)*, 222
summary,SimLabels-method *(*ParameterSet*)*, 84
summary,SimLnorm-method *(*VirtualDist-class*)*, 222
summary,SimLogis-method *(*VirtualDist-class*)*, 222

summary,SimMatrix-method (SimMatrix-class), 158
summary,SimMissing-method (SimMissing-class), 161
summary,SimMisspec-method (SimMisspec-class), 162
summary,SimModel-method (SimModel-class), 170
summary,SimModelOut-method (SimModelOut-class), 173
summary,SimNbinom-method (VirtualDist-class), 222
summary,SimNorm-method (VirtualDist-class), 222
summary,SimParam-method (SimParam-class), 176
summary,SimPois-method (VirtualDist-class), 222
summary,SimREqualCon-method (SimREqualCon-class), 183
summary,SimResult-method (SimResult-class), 186
summary,SimResultParam-method (SimResultParam-class), 189
summary,SimRSet-method (ParameterSet), 84
summary,SimSet-method (SimSet-class), 190
summary,SimT-method (VirtualDist-class), 222
summary,SimUnif-method (VirtualDist-class), 222
summary,SimVector-method (SimVector-class), 202
summary,SimWeibull-method (VirtualDist-class), 222
summary,SymMatrix-method (SymMatrix-class), 216
summary,VirtualRSet-method (ParameterSet), 84
summaryFit, 208
summaryFit,ANY-method (summaryFit), 208
summaryFit,SimResult-method (summaryFit), 208
summaryFit,SimResultParam-method (summaryFit), 208
summaryFit-methods (summaryFit), 208
summaryMisspec, 210
summaryMisspec,ANY-method (summaryMisspec), 210
summaryMisspec,SimResultParam-method (summaryMisspec), 210
summaryMisspec-methods (summaryMisspec), 210
summaryParam, *172*, *174*, *187*, 211
summaryParam,ANY-method (summaryParam), 211
summaryParam,SimModelMIOut-method (summaryParam), 211
summaryParam,SimModelOut-method (summaryParam), 211
summaryParam,SimResult-method (summaryParam), 211
summaryParam,SimResultParam-method (SimResultParam-class), 189
summaryParam-methods (summaryParam), 211
summaryPopulation, *143*, *174*, *187*, 213
summaryPopulation,ANY-method (summaryPopulation), 213
summaryPopulation,SimDataOut-method (SimDataOut-class), 143
summaryPopulation,SimModelOut-method (SimModelOut-class), 173
summaryPopulation,SimResult-method (SimResult-class), 186
summaryPopulation-methods (summaryPopulation), 213
summaryShort, *158*, *202*, 214, *223*
summaryShort,ANY-method (summaryShort), 214
summaryShort,matrix-method (summaryShort), 214
summaryShort,SimBeta-method (VirtualDist-class), 222
summaryShort,SimBinom-method (VirtualDist-class), 222
summaryShort,SimCauchy-method (VirtualDist-class), 222
summaryShort,SimChisq-method (VirtualDist-class), 222
summaryShort,SimExp-method (VirtualDist-class), 222
summaryShort,SimF-method (VirtualDist-class), 222
summaryShort,SimGamma-method (VirtualDist-class), 222
summaryShort,SimGeom-method (VirtualDist-class), 222
summaryShort,SimHyper-method (VirtualDist-class), 222
summaryShort,SimLnorm-method (VirtualDist-class), 222
summaryShort,SimLogis-method (VirtualDist-class), 222

summaryShort,SimMatrix-method
(SimMatrix-class), 158
summaryShort,SimNbinom-method
(VirtualDist-class), 222
summaryShort,SimNorm-method
(VirtualDist-class), 222
summaryShort,SimPois-method
(VirtualDist-class), 222
summaryShort,SimT-method
(VirtualDist-class), 222
summaryShort,SimUnif-method
(VirtualDist-class), 222
summaryShort,SimVector-method
(SimVector-class), 202
summaryShort,SimWeibull-method
(VirtualDist-class), 222
summaryShort,vector-method
(summaryShort), 214
summaryShort-methods (summaryShort), 214
SymMatrix, *5*, *6*, *69*, *82*, *123*, *159*, *162*, *193–195*, *197*, *198*, *202*
symMatrix, *157*, *202*, 215, *216*, *222*, *224*
SymMatrix-class, 216

tagHeaders, 217
tagHeaders,ANY-method (tagHeaders), 217
tagHeaders,VirtualRSet-method
(tagHeaders), 217
tagHeaders-methods (tagHeaders), 217
toFunction, 218, *223*
toFunction,ANY-method (toFunction), 218
toFunction,SimBeta-method
(VirtualDist-class), 222
toFunction,SimBinom-method
(VirtualDist-class), 222
toFunction,SimCauchy-method
(VirtualDist-class), 222
toFunction,SimChisq-method
(VirtualDist-class), 222
toFunction,SimExp-method
(VirtualDist-class), 222
toFunction,SimF-method
(VirtualDist-class), 222
toFunction,SimGamma-method
(VirtualDist-class), 222
toFunction,SimGeom-method
(VirtualDist-class), 222
toFunction,SimHyper-method
(VirtualDist-class), 222
toFunction,SimLnorm-method
(VirtualDist-class), 222
toFunction,SimLogis-method
(VirtualDist-class), 222
toFunction,SimNbinom-method
(VirtualDist-class), 222
toFunction,SimNorm-method
(VirtualDist-class), 222
toFunction,SimPois-method
(VirtualDist-class), 222
toFunction,SimT-method
(VirtualDist-class), 222
toFunction,SimUnif-method
(VirtualDist-class), 222
toFunction,SimWeibull-method
(VirtualDist-class), 222
toFunction-methods (toFunction), 218
toSimSet, 218
toSimSet,ANY-method (toSimSet), 218
toSimSet,SimModelOut-method (toSimSet),
218
toSimSet,SimRSet-method (toSimSet), 218
toSimSet-methods (toSimSet), 218
twoTailedPValue, 219

validateCovariance, 220
validateObject, 220
validatePath, 221
vectorizeObject, 221
vectorizeObject,ANY,ANY-method
(vectorizeObject), 221
vectorizeObject,matrix,matrix-method
(vectorizeObject), 221
vectorizeObject,MatrixSet,SimGenLabels-method
(vectorizeObject), 221
vectorizeObject,vector,vector-method
(vectorizeObject), 221
vectorizeObject,VirtualRSet,SimLabels-method
(vectorizeObject), 221
vectorizeObject-methods
(vectorizeObject), 221
VirtualDist, *91*, *134–136*, *140*, *147*, *148*, *152*, *154–157*, *164*, *175*, *176*, *182*, *184*, *191*, *200*, *201*, *203*, *215*, *218*
VirtualDist-class, 222
VirtualRSet, *8*, *11*, *83*
VirtualRSet-class (ParameterSet), 84

weightedMean, 225
whichMonotonic, 225
writeLavaanCode, 226
writeLavaanConstraint, 227
writeLavaanIndividualConstraint, 227
writeLavaanNullCode, 228